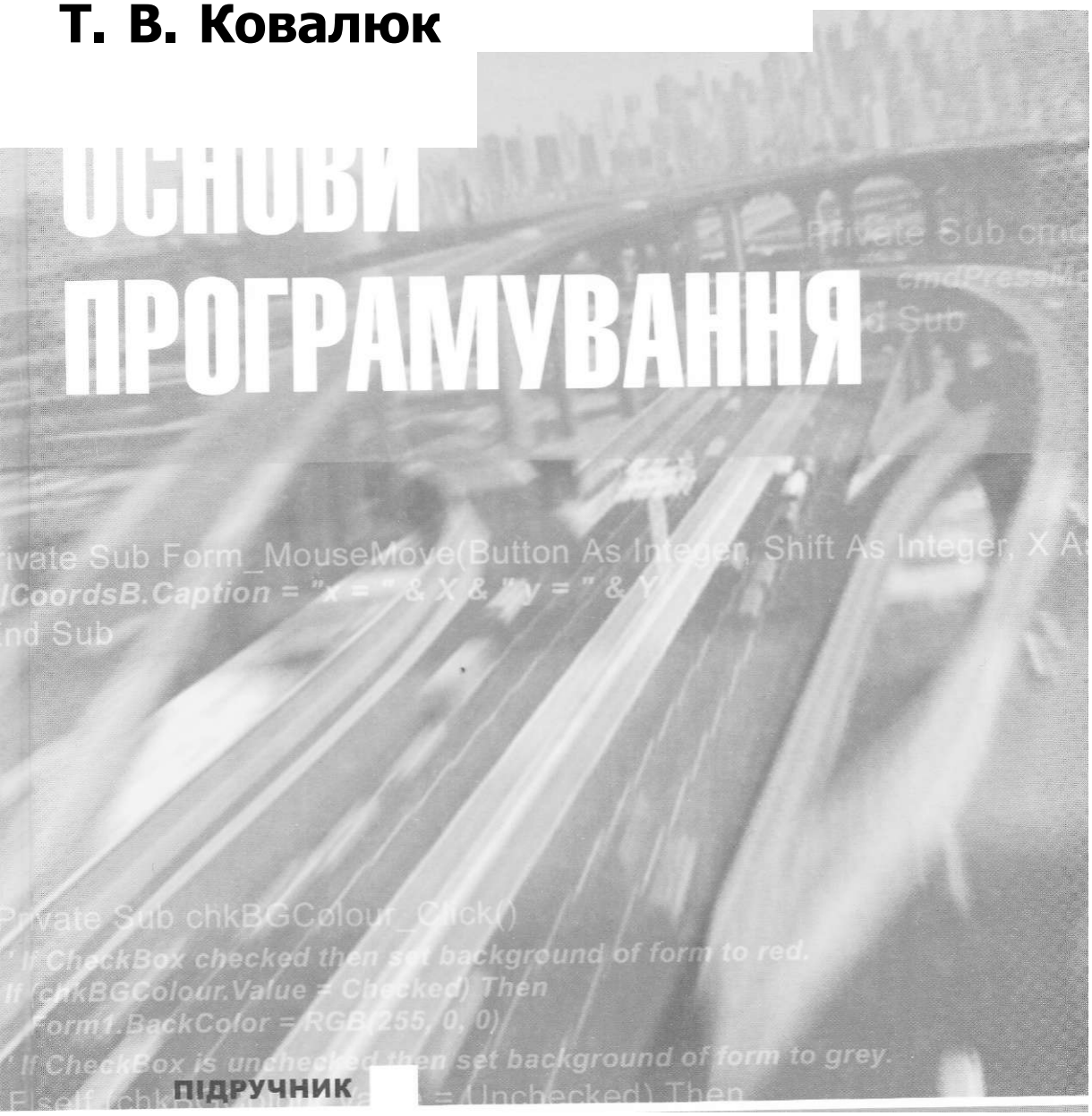
інформатика

Т. В. Ковалюк

ОСНОВИ ПРОГРАМУВАННЯ



ПІДРУЧНИК

ДЛЯ ВИЩИХ НАВЧАЛЬНИХ ЗАКЛАДІВ

Т. В. КОВАЛЮК

ОСНОВИ ПРОГРАМУВАННЯ

Затверджено Міністерством освіти і науки України як підручник для студентів вищих навчальних закладів, які навчаються за напрямами «Комп'ютерні науки», «Комп'ютеризовані системи, автоматика і управління», «Комп'ютерна інженерія», «Прикладна математика»

Серія «ІНФОРМАТИКА»
За загальною редакцією академіка НАН України
М. З.Згуровського

Київ
Видавнича група ВНУ
2005

ББК 32.973-018я73
К-56
УДК 004.42(075.8)

Рецензенти: Г. О. Цейтлін, доктор технічних наук, професор, завідувач кафедри програмного забезпечення автоматизованих систем Міжнародного Соломонова університету; В. М. Синєглазов, доктор технічних наук, професор, директор Інституту електроніки та систем управління Національного авіаційного університету.

Гриф надано Міністерством освіти і науки України, лист № 14/18.2-1564 від 08.07.2004 р.

Ковалюк Т. В.

К-56 Основи програмування. — К: Видавнича група ВНУ, 2005. — 384 с.: іл.
ІЗВИ 966-552-138-1

Книжка є досить повним підручником з класичних методів програмування. У ній розглядаються основні поняття алгоритмізації, а також техніка застосування у програмуванні базових алгоритмічних структур і структур даних. У підручнику розв'язана велика кількість різноманітних задач з обробки масивів і файлів, сортування та пошуку, створення і обробки графічних зображень тощо. Усі розв'язання супроводжуються детальними коментарями, описом алгоритмів та застосованої техніки програмування. Велика увага приділяється важливим алгоритмам матричної та векторної алгебри, обробці динамічних структур даних і обчисленням на графах. Окремий розділ книги присвячено теоретичним засадам структурного та об'єктно-орієнтованого програмування. Як робоча мова програмування у підручнику використовується Разсаі.

Книжка призначена для студентів, які навчаються за напрямом «Комп'ютерні науки» і вивчають сучасні інформаційні технології в рамках дисципліни «Основи програмування та алгоритмічні .мови», а також для викладачів зазначеної дисципліни.

ББК 32.973-018я73

Серія підручників «Інформатика» виходить у світ за підтримки видавництв «Издательский дом "Питер"», м. Санкт-Петербург та «Освітня книга», м. Київ.

І5ВИ 966-552-138-1

Стислий **зміст**

Зміст

Частина 2
Організація даних

Передмова

Підручник, що пропонується до уваги читача, є ґрунтовним і водночас досить легким для сприйняття посібником з класичних методів програмування. Зміст підручника відповідає навчальній програмі з дисципліни «Основи програмування та алгоритмічні мови», за якою проводиться підготовка бакалаврів за напрямом «Комп'ютерні науки» у вищих навчальних закладах України.

Підручник розрахований на початківців, тобто не потребує від читача попередньої підготовки. Проте засвоїти викладений матеріал буде легше, якщо студент володітиме знаннями і вміннями з програмування в межах, передбачених шкільним курсом основ інформатики та обчислювальної техніки. Взагалі до лав читацької аудиторії можуть увійти студенти початкових курсів з будь-яких спеціальностей.

Робочою мовою програмування для даної книжки обрано PascaI, оскільки її виразні засоби вдало поєднують такі риси, як простота і потужність. На думку переважної більшості викладачів і програмістів, PascaI вже понад 20 років залишається найкращою навчальною мовою програмування. Розглядаючи досить детально синтаксис та семантику значної кількості конструкцій цієї мови, автор не ставить собі за мету описати всі її тонкощі й особливості. Основна увага приділяється вивченню методології побудови програм, а мова програмування використовується лише як засіб для ілюстрації концепцій, методів та прийомів, якими має оволодіти програміст.

Підручник складається з одинадцяти розділів, які за тематикою об'єднані в дві частини. Перша частина, «Організація програм», містить теоретичний і практичний матеріал з таких питань, як структура комп'ютерів, основи алгоритмізації, поняття типів даних, змінних, операцій та операторів, зображення даних в пам'яті комп'ютера, організація розгалужених та циклічних програм, поняття підпрограм та їх застосування. Окремі розділи першої частини посібника присвячені програмуванню графіки та теоретичним основам методології структурного, модульного й об'єктно-орієнтованого програмування. У другій частині підручника, що має назву «Організація даних», розглядаються фундаментальні структури даних, зокрема масиви, рядки, записи, множини, файли, наведено приклади найважливіших обчислювальних алгоритмів, де застосовуються ці структури. Крім того, обговорюються питання організації динамічних структур даних та досліджуються алгоритми на графах. Цей матеріал стане у нагоді в першу чергу для студентів, які паралельно з дисципліною «Основи програмування та алгоритмічні мови» вивчають ще й дисципліну «Основи дискретної математики».

Велику увагу автор підручника приділяє практичним аспектам розробки програм. Всі теоретичні положення з алгоритмізації та програмування ілюструються численною кількістю алгоритмів та їх програмних реалізацій. Приклади підібрані так, щоб продемонструвати якомога більше прийомів і методів структурного програмування. Всі наведені у підручнику програми супроводжуються детальним

описом використаних у них алгоритмів і даних. Майже до всіх операторів в текстах програм додано коментарі.

У кожному розділі підручника наведені контрольні запитання, вправи для самоперевірки та задачі для самостійного виконання. Задачі мають різний рівень складності: деякі з них вимагають від студента лише ґрунтовного засвоєння матеріалу розділу, а деякі — наявності навиків творчого мислення і вміння знаходити нетривіальні рішення. В кінці книжки наведено 15 задач підвищеної складності, які пропонувалися протягом кількох останніх років на різних олімпіадах з програмування в Україні.

Під час роботи над книгою автору допомагали багато людей. Вони давали цінні поради, рецензували книгу або окремі її розділи, редагували матеріал. Автор вдячний всім, хто сприяв підвищенню якості книги: рецензентам Г. Є. Цейтліну та В. М. Синєглазову, науковим редакторам І. О. Завадському та В. П. Паську, директорам Видавничої групи ВНУ О. В. Полякову та І. В. Стеценко, а також студентам факультету інформатики та обчислювальної техніки НТУУ «КПІ», на яких було апробовано методику викладання матеріалу і які розробляли та тестували всі приклади, що наведені в підручнику.

Від видавництва

Коди програм, процес створення яких описаний у даному підручнику, можна знайти в мережі Інтернет за адресою Нйрі/Умллм/.ОБУіІа.іпГо. Доступ до кожного файла з текстом програми здійснюється через посилання на відповідний розділ книжки.

Інформацію про книжки Видавничої групи ВНУ ви можете знайти на сайті

Час™иа_1

Організація програм

Розділ 1

Основні поняття та означення

- Основні принципи побудови обчислювальних систем
- Архітектура системи команд і форми зображення даних
- Системи числення
- Етапи створення програм – від аналізу поставленої задачі до налагодження
- Поняття алгоритму й елементарні алгоритмічні структури

1.1. Поняття архітектури комп'ютера

Поняття архітектури обчислювальних систем є одним з основних в інформатиці. Уперше термін «архітектура комп'ютера» був введений фірмою IBM при розробці обчислювальних систем серії IBM 360 і застосований до тих засобів, які може використовувати програміст під час написання програм на рівні машинних команд.

Під цим терміном розуміли структуру комп'ютера з погляду програміста, так звану логічну архітектуру, що є поняттям архітектури у вузькому розумінні. Воно охоплює формати команд, форми зображення даних, методи адресації й управління операціями введення-виведення. При цьому з розгляду випадають такі аспекти, як фізична організація пристроїв комп'ютера, ієрархія пам'яті, методи паралельної обробки інформації, а також багато-багато інших, які часто узагальнюються терміном «організація комп'ютера» чи структурна організація комп'ютера.

Далі ми використовуватимемо термін «архітектура», розуміючи під цим поняттям логічну структуру у сукупності з фізичною структурною організацією.

В основу архітектури більшості сучасних комп'ютерів покладено принципи, які ще 1946 року були сформульовані у звіті «Попереднє обговорення логічного конструювання електронного обчислювального пристрою» Джоном фон Нейманом і його колегами Г. Голдстайном та А. Берксом. Усі комп'ютери, побудовані згідно 3 цими принципами, тепер відомі як комп'ютери з фоннеймановською архітектурою.

Основні принципи архітектури фон Неймана такі:

- використання двійкової системи числення для кодування інформації у комп'ютері;

4- програмне керування роботою комп'ютера;

4 збереження програм у пам'яті комп'ютера;

4 адресація пам'яті.

1.1.1, Принцип використання двійкової системи числення

Інформація (команди і дані) у комп'ютері кодуються у *двійковій системі числення. Система числення* — це система позначення чисел. У повсякденній практиці використовується десяткова система, в якій числа записуються за допомогою десяти арабських цифр 0, 1,2,..., 9. У двійковій системі числення є тільки дві цифри: 0 та 1. Зручність використання двійкової системи в обчислювальній техніці обумовлена тим, що електронні перемикачі можуть перебувати тільки в одному із двох станів: увімкненому чи вимкненому. Ці стани можна кодувати двома цифрами: одиницею або нулем. Так само в двох станах у кожен окремий момент часу може перебувати канал передачі даних: «рівень напруги високий» або «рівень напруги низький». Крім того, логіка висловлень є двійковою логікою: будь-яке висловлення в кожен момент є істинним або хибним. Якщо висловлення є істинним, йому відповідає значення 1, якщо хибне — значення 0.

Одна двійкова цифра називається *бітом* (від англ. йгпагу сіі^іг - двійкова цифра). За допомогою одного біта можна закодувати два інформаційних повідомлення, що умовно позначаються символами «0» та «1». Із двох бітів можна утворити коди чотирьох інформаційних повідомлень: «00», «01», «10» та «11»; із трьох бітів — восьми. У загальному випадку за допомогою *n* бітів можна закодувати 2" інформаційних повідомлень.

Послідовність, що складається з восьми бітів, у сучасній обчислювальній техніці прийнято називати *байтом.* За допомогою одного байта можна закодувати 2^8 = 256 різних повідомлень і зобразити, наприклад, значення цілих чисел від 0 до 255. Ці самі повідомлення можна розглядати і як числа від -128 до 127 (їх кількість теж дорівнює 256), символи, логічні значення тощо.

Байт є одиницею обсягу пам'яті. Для позначення тисяч та мільйонів байтів використовуються такі одиниці, як кілобайт (1 Кбайт = 2^{10} = 1 024 байт), мегабайт (1 Мбайт = 2^{20} = 1 048 576 байт) і гігабайт (1 Гбайт = 2^{30} = 1 073 741 824 байт).

Послідовностями двійкових цифр можна кодувати не лише числа, а й довільні інформаційні повідомлення. Зокрема, загальноприйнята система кодування A5CII ставить коди у відповідність 256 символам. Наприклад, латинській літері «а» відповідає код 97_{10}= 1100001_2 , а символу «@» - код 64_{10}= 1000000_2. Складніші системи кодування дають можливість закодувати зображення, звук тощо.

1.1.2. Принцип програмного керування роботою комп'ютера

Сучасний комп'ютер — це сукупність технічних і програмних засобів, які призначені для автоматизованої обробки дискретних даних відповідно до заданого алгоритму. Алгоритм — це основне поняття математики та обчислювальної техніки і згідно зі стандартом 150 2382/1-84 він визначається як «скінченний набір інструкцій, які описують процес розв'язування задачі за допомогою скінченної кількості операцій».

Усі обчислення, що виконує комп'ютер, мають бути зображені у пам'яті у вигляді програми, складеної з інструкцій, які прийнято називати командами. Кожна

команда вказує на певну операцію, яку має виконати комп'ютер. Сукупність команд, що їх використовує конкретний комп'ютер, називається системою команд.

Програма складається з послідовності команд, які процесор виконує автоматично в певному порядку. Змінювати порядок виконання команд залежно від проміжних результатів обчислень дозволяють команди умовного та безумовного переходів. Завдяки цьому можна багаторазово виконувати одну й ту саму послідовність команд програми, а отже, набагато скоротити її розміри.

1.1.3. Принцип збереження програм у пам'яті комп'ютера

Цей один із найважливіших принципів, який називають також принципом однорідності пам'яті, полягає в тому, що команди та дані зберігаються в одних і тих самих областях пам'яті і нерідко кодуються тими самими станами комірок пам'яті. Комп'ютер обробляє і команди, і дані однаково, тобто над командою можна виконати ті самі операції, що й над будь-якими даними. Це розкриває нові можливості для перетворення програми безпосередньо при її виконанні. Наприклад, команди однієї частини програми можна отримати як результат виконання команд іншої. Ця можливість покладена в основу трансляторів, які перетворюють текст програми на мові високого рівня у послідовність команд конкретного комп'ютера.

Принцип однорідності пам'яті, запропонований у статті фон Неймана, використовувався в обчислювальних системах, які створювались у Прінстонському університеті, тому архітектура таких обчислювальних систем дістала назву *прінсгпонської.* Практично водночас у Гарвардському університеті запропонували іншу архітектуру, згідно з якою команди і дані повинні використовувати окрему пам'ять. Прінстонська архітектура була і є домінуючою, однак останнім часом завдяки широкому використанню нових технологій пам'яті все більше поширюються комп'ютери з гарвардською архітектурою.

1.1.4. Принцип адресності пам'яті

Для виконання програми необхідно, щоб команди та дані перебували в основній пам'яті. Основна пам'ять — це послідовність комірок, кожна з яких має свій номер — *адресу.* Розмір комірки зазвичай дорівнює восьми двійковим розрядам — байту. Числове значення у пам'яті, як правило, займає декілька сусідніх байтів. Для збереження цілих чисел найчастіше використовують один, два або чотири байти, для нецілих (дійсних) - 4, 6, 8 або 10 байт.

Адресою числа вважається адреса першого байта області пам'яті, відведеної для збереження числа. Так, якщо 32-розрядне двійкове число зберігається в комірках 200, 201, 202 та 203, то його адресою буде 200. Такий метод адресації називається адресацією молодшого байта і використовується в комп'ютерах фірми Іпгеі і міні- комп'ютерах компанії БЕС. Існує й інший метод, коли адресою числа є його старший байт (метод адресації старшого байта). Такий метод використовується в комп'ютерах фірм ІВМ та Моїогоіа.

Процесор у будь-який момент може отримати доступ до будь-якої комірки пам'яті. Така пам'ять називається пам'яттю з довільним доступом.

1.2. Архітектура комп'ютерів фон Неймана

У класичній роботі фон Неймана перелічено основні пристрої, з використанням яких може бути побудований комп'ютер. Вважають, що комп'ютери такого типу мають *фоннеймановську архітектуру.* Типова фоннеймановська обчислювальна машина має такі складові: арифметико-логічний пристрій, пристрій керування, пам'ять і пристрої введення-виведення (рис. 1.1).

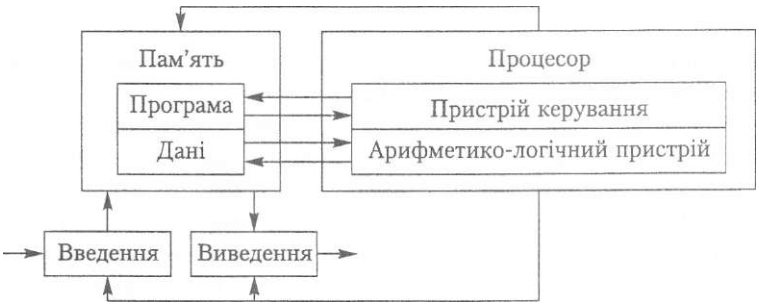


Рис. 1.1. Структура фоннеймановської обчислювальної машини

В обчислювальних машинах із класичною фоннеймановською архітектурою функції обробки інформації покладені на *арифметико-логічний пристрій* (АЛП). До функцій АЛП входить у першу чергу виконання арифметичних і логічних команд та команд зсуву. Отже, цей пристрій забезпечує обробку вхідних даних і формування результату. АЛП - це не один, а ціла група операційних пристроїв, кожен з яких реалізує певну підмножину операцій.

Пристрій керування (ПК) — найголовніша частина комп'ютера, що координує роботу всіх його пристроїв та забезпечує виконання всіх програм. Основною функцією цього пристрою є формування сигналів, необхідних для вибирання команд із пам'яті в порядку, що задається програмою, і подальше виконання цих команд. Крім того, ПК формує сигнали, необхідні для синхронізації та координації дій зовнішніх і внутрішніх пристроїв комп'ютера. Синхронізуючі сигнали - це сигнали, що визначають, коли має бути виконана певна операція. Реальні синхронізуючі сигнали, які керують пересиланням даних і виконанням команд, генеруються схемами керування.

Пристрій керування можна уявити собі як окремий блок, котрий взаємодіє з іншими блоками комп'ютера. Однак на практиці так буває рідко. Більша частина керуючих схем фізично розподілена між різними компонентами комп'ютера: процесором, чипсетом, контролерами введення-виведення та шин.

АЛП і ПК дуже тісно взаємодіють між собою, і їх часто реалізують єдиним пристроєм, відомим як центральний процесор, чи просто процесор. Процесор - це основа будь-якого комп'ютера, і швидкість роботи комп'ютера цілковито залежить від моделі та параметрів процесора.

Крім результуючих даних, АЛП формує також ознаку, яка визначає коректність отриманого результату, а також інші його характеристики (рівність нулю,

переповнення, парність і т. ін.). Значення ознаки результату може аналізуватися ПК для прийняття рішення стосовно подальшої послідовності виконання програми.

Пам'ять комп'ютера призначена для збереження інформації й оперативного обміну нею з іншими компонентами комп'ютера. Пам'ять поділяють на внутрішню та зовнішню. *Внутрішня пам'ять* складається з регістрів процесора, основної пам'яті та кеш-пам'яті.

Регістри процесора — це найбільш швидкодіючий, але найменший за обсягом різновид пам'яті комп'ютера. Зазвичай регістрів у процесорі небагато, і тільки в комп'ютерах з неповним набором команд (Кесіисесі ІпзСгисгіоп Зеї Сошриїег, КІ5С) їх кількість може сягати кількох сотень. Регістри залежно від свого призначення можуть мати обсяг від 1 до 10 байт.

Основна пам'ять може включати пам'ять двох типів — *постійну* й *оперативну.*

Постійна пам'ять (Кеасі Опіу Метогу, ЕОМ) реалізується у вигляді постійного запам'ятовуючого пристрою, дані в якому не можна модифікувати, їх можна тільки читати. Основне призначення КОМ — підтримання процедур початкового завантаження операційної системи та обслуговування переривань, які припиняють виконання програми з метою реалізації спеціальних системних дій.

Процес зчитування інформації з постійної пам'яті майже не відрізняється від процесу зчитування з оперативної пам'яті. З іншого боку процес записування інформації в КОМ набагато складніший і потребує великих затрат часу й енергії. Записування інформації в постійну пам'ять називають її програмуванням. Сучасні пристрої постійної пам'яті - це мікросхеми, інформація до яких може записуватись як при виготовленні, так і після нього.

Оперативна пам'ять (Капкіопі Ассезз Метогу, К.АМ) використовується і для читання, і для запису інформації. Під час роботи комп'ютера в ній зберігаються програми та дані. Оперативна пам'ять комп'ютера організована у вигляді множини байтів, або комірок, у яких зберігаються числові та символьні значення. Байти оперативної пам'яті послідовно пронумеровані починаючи з 0. Ці номери байтів є їх *адресами.* Залежно від типів даних, над якими виконуються дії, байти групуються у слова (два байти) або подвійні слова (чотири байти).

Число, що зберігається в комірці, — це її *значення,* або *вміст.* Якщо в *к*-й комірці міститься, наприклад, число *т,* то прийнято говорити «вміст комірки з адресою *к* дорівнює *т».* Під час читання даних з оперативної пам'яті вміст комірки не змінюється. Для його зміни потрібно записати в цю комірку нове значення, або, інакше кажучи, *присвоїти* комірці нове значення. Поняттю адреси оперативної пам'яті повною мірою відповідає поняття змінної в алгебрі. Змінну позначають ідентифікатором. Цьому ідентифікатору ставиться у відповідність адреса комірки оперативної пам'яті. Якщо змінній надати певне значення, то воно записується за адресою відповідної комірки пам'яті.

Особливістю КАМ є неможливість зберігання інформації в ній після вимикання живлення комп'ютера. Саме цим вона відрізняється від пам'яті КОМ, в якій дані зберігаються незалежно від наявності живлення.

Між регістрами та процесором часто розміщується *кеш-пам'ять,* призначена для узгодження швидкостей роботи основної пам'яті та процесора. Сучасні ком-

п'ютери мають декілька рівнів кеш-пам'яті, які позначаються буквою Ь з певним номером. У більшості персональних комп'ютерів кеш-пам'ять має два рівні — Ь1 та Ь2, в останніх розробках все частіше з'являється кеш-пам'ять третього рівня, а в проектах навіть і четвертого. Обсяг кеш-пам'яті вимірюється десятками і сотнями кілобайтів, а оперативної - десятками і сотнями мегабайтів. Оперативна пам'ять сучасних персональних комп'ютерів сягає кількох гігабайтів.

Для довгострокового збереження великого обсягу даних і програм потрібна *зовнішня пам'ять.* Зокрема, у зовнішній пам'яті зберігається все програмне забезпечення комп'ютера (системне та прикладне). До складу зовнішньої пам'яті входять різноманітні пристрої: накопичувачі на жорстких і гнучких магнітних дисках, пристрої на касетній магнітній стрічці (стримери), накопичувачі на оптичних лисках СО-КОМ та СВ-КЛ¥ (від англ. Сошрасї Вікк Кеасі Опіу Метогу - компакт- диск тільки для читання, Сошрасї Бізк Кеасі \¥гіїе Метогу — компакт-диск для запису та читання відповідно) тощо. Накопичувачі на жорстких магнітних дисках відрізняються від накопичувачів на гнучких магнітних дисках за конструкцією, обсягом даних, що зберігаються, а також часом пошуку, запису і зчитування інформації. Необхідно пам'ятати, що інформація, яка зберігається у зовнішній пам'яті, стане доступною для процесора тільки після того, як буде переписана в основну пам'ять.

Накопичувач можна розглядати як пристрій, у якому носій інформації поєднаний з приводом, що є механізмом зчитування-запису інформації, та відповід- - им керуючим пристроєм. Дисковий привід називається дисководом, а стрічковий - стримером. Комп'ютери зазвичай мають декілька дисководів для роботи з дисками різних типів.

Пристрої введення-виведення є важливою складовою будь-якого комп'ютера. Зони забезпечують взаємодію комп'ютера з навколишнім середовищем, користувачами, об'єктами керування та іншими комп'ютерами.

Зовнішні пристрої за їх призначенням можна розділити на два основних різновиди - для тривалого зберігання інформації та для взаємодії комп'ютера із зовнішнім середовищем, а саме користувачами, іншими комп'ютерами й об'єктами керування.

Серед них можна виділити пристрої введення (клавіатура, сканер, графічний лланшет, маніпулятори типу «миша», джойстик, світлове перо тощо), виведення лринтер, плотер), діалогові засоби користувача (дисплеї, пристрої мовного введення-виведення - мікрофонні акустичні системи, синтезатори звуку тощо), засоби зв'язку та телекомунікацій (мережні плати, модеми).

Більшість зовнішніх пристроїв мають свої процесори (контролери), які є простішими за центральний процесор і виконують інші набори команд. Контролери можуть переносити дані із зовнішніх носіїв до оперативної пам'яті (читання, або зведення із «зовнішнього світу») чи навпаки (запис, або виведення даних у «зовнішній світ»). Кожному пристрою введення-виведення виділено окрему ділянку оперативної пам'яті - порт. їз нього пристрій бере дані для зовнішнього носія, записуючи їх, наприклад, на диск або на екран комп'ютера. І саме в порт записуються дані, що надходять з клавіатури або -дисковода. -- — -

Сигнали синхронізації дій усіх пристроїв передаються по керуючих лініях — шинах. Шини комп'ютера повинні забезпечити передачу інформації між:

- процесором та оперативною пам'яттю (шина процесор-пам'ять);
- процесором і портами введення-виведення зовнішніх пристроїв;
- оперативною пам'яттю і портами введення-виведення зовнішніх пристроїв у режимі прямого доступу до пам'яті.

Шина процесор-пам'ять забезпечує безпосередній зв'язок між процесором комп'ютера та основною пам'яттю. У сучасних комп'ютерах таку шину іноді називають шиною переднього плану і позначають як P8B (від англ. Fronï-Sicie Виз). Інтенсивний обмін даними між процесором та пам'яттю вимагає, щоб кількість інформації, яка передається за одиницю часу (секунду) цією шиною (пропускна спроможність шини), була якомога більшою. Від пропускної спроможності шини процесор-пам'ять значною мірою залежить продуктивність комп'ютерів із фоннеймановською архітектурою. Функції різних шин конструктори іноді покладають на єдину системну шину, але з погляду швидкодії краще, коли дані між процесором і пам'яттю передаються окремою шиною.

Зв'язок процесора чи пам'яті із пристроями введення-виведення забезпечують *шини введення-виведення.* На відміну від шини процесор-пам'ять, такі шини містять менше ліній, але фізична довжина цих ліній може бути більшою. Велика кількість і різноманітність зовнішніх пристроїв у різних типах комп'ютерів обумовили необхідність розроблення стандартів для таких шин. Стандартизація шин дозволяє розробникам зовнішніх пристроїв працювати незалежно, а користувачам — самостійно формувати потрібну конфігурацію комп'ютера.

Фізично шини складаються з великої кількості паралельних металевих провідників — ліній, розміщених на системній платі чи кристалі мікросхеми. Серед ліній будь-якої шини можна виділити три функціональні групи: *шина адреси, шина даних* і *шина керування.*

Шиною адреси передаються адреси комірок пам'яті, номери регістрів процесора, адреси портів введення-виведенні і т. ін. Кількість ліній, виділених для передачі адреси, становить ширину шини адреси і визначає максимально можливий обсяг пам'яті комп'ютера, який може адресуватися.

Лінії, якими дані (команди чи операнди) передаються між блоками системи, називаються шиною даних. Найважливіші параметри шини даних - ширина та пропускна спроможність. Ширина шини даних вказує на кількість бітів інформації, які можуть бути передані по шині за один її цикл.

Використання окремих шин для адрес і даних дає можливість значно підвищити продуктивність комп'ютера, особливо під час записування інформації в пам'ять - адреса комірки пам'яті та дані, що записуються, передаються паралельно.

Поряд із лініями для передачі адрес та даних обов'язковим атрибутом будь-якої шини є лінії, якими передається керуюча інформація та інформація про стан пристроїв введення-виведення. Сукупність таких ліній прийнято називати шиною керування. Цією шиною передаються сигнали синхронізації, переривання, арбітра шини тощо.

Загалом, функціонування фоннеймановського комп'ютера можна описати так:

- комп'ютер за допомогою пристроїв введення приймає інформацію у вигляді програм та даних і записує її в пам'ять;
- інформація, що зберігається в пам'яті, під керуванням програми шинами пересилається в арифметико-логічний пристрій для подальшої обробки;
- дані, отримані після обробки інформації, спрямовуються (також шинами) на пристрої виведення.

1.3. Архітектура системи команд

Щоб виконати операції над даними будь-якого типу, в оперативній пам'яті потрібно розмістити відповідні команди програми. Коди команд, так само як і їх структура, розробляються під час проектування комп'ютера. Повний перелік команд, які здатний виконати даний комп'ютер, називається його *системою команд.* Система команд сучасного комп'ютера може містити сотні і навіть тисячі інструкцій. Типова команда комп'ютера фоннеймановської архітектури у загальному вигадку має задавати:

- операцію, яку слід виконати;
- адреси даних (операндів), над якими має бути виконана операція;
- адресу пам'яті, де потрібно зберегти результат виконання операції.

Отже, команда складається з двох частин - командної й адресної. У даному газі термін «адресність» означає кількість операндів, вказаних в адресній частині :оманди. Триадресна команда може мати таку структуру, як показано на рис. 1.2. ХОП - код операції, А1, А2, А3 - адреси комірок оперативної пам'яті, в яких містяться дані та результат виконання операції.

к о п	А1	А2	А3

Рис. 1.2. Спрощена структура триадресної команди

Як правило, команда розміщується в декількох байтах. Адреса команди визначається як адреса її першого байта. Від формату команди залежить її структура, тобто кількість двійкових розрядів, які відводяться для збереження команди в пам'яті, а також кількість та розташування окремих груп розрядів команди. Код :перації та адреси комірок даних — це послідовності двійкових розрядів.

Команди оперують даними, які прийнято називати операндами. Базовими тилами операндів є адреси, числа, символи та логічні значення. Крім базових типів комп'ютер здатний обробляти і складніші інформаційні структури - графічні з збраження, аудіо-, відеоінформацію, анімацію. Така інформація є похідною від : азових типів і зберігається, як правило, на зовнішніх носіях.

Наведена триадресна команда (рис. 1.2) відповідає зображенню звичайних алгебричних операцій вигляду $x = y$ х $г$, що читається так: «виконати операцію

множення над змінними *у* і *г,* результат присвоїти змінній х» або «виконати операцію множення над вмістом комірок *у* і *г,* результат записати до комірки з адресою *х».*

Команди програми виконуються послідовно, починаючи з першої. Це можна реалізувати завдяки тому, що адреси комірок оперативної пам'яті, в яких розміщено команди програми, становлять послідовність. Процесор має *регістр команд,* призначений для їх автоматичного виконання, та *регістр-лічильник команд,* у якому зберігається адреса наступної виконуваної команди. Для виконання наступної команди вміст лічильника команд необхідно збільшити на довжину поточної команди (визначається в байтах), з тим щоб отримати адресу наступної команди. З метою автоматичного виконання команди потрібно вибрати її за вказаною адресою і передати до регістра команд. Щоб зв'язати адреси комірок оперативної пам'яті з ідентифікаторами змінних, які використовуються у процесі розв'язання конкретної задачі, створюється таблиця адрес. Процес її упорядкування називається *розподілом пам'яті.*

Приклад 1.1

Нехай потрібно обчислити значення змінної *а,* заданої виразом *а = (Ь + с) х сі,* за умови, що пам'ять розподіляється, починаючи від комірки з адресою 100. Тоді пам'ять, призначену для зберігання змінних, можна розподілити, скажімо, так: змінній а поставити у відповідність комірку з адресою 100, змінній *Ь* — комірку з адресою 101 тощо (табл. 1.1); змінна *г* призначена для збереження проміжних результатів.

Таблице 1.1. Приклад розподілу пам'яті під змінні

Ідентифікатор змінної	Адреса комірки оперативної пам'яті
а	**100**
Ь	**101**
с	**102**
(і	**103**
г	**106**

Запишемо машинну програму обчислення значення змінної *а,* заданої виразом *а = (Ь + с)* х *а. У* цій програмі операція введення задається кодом 01, операція виведення - кодом 02, операція додавання - кодом 03, операція множення - кодом- 05 (табл. 1.2),

Таблиця 1.2. Машинна програма обчислення виразу а = *(Ь + с)* х *б*

кап	А1	А2	А3	Примітка
01	I01	000	000	Увести значення змінної *Ь* до комірки з адресою 101
01	•'.'02	000	000	Увести значення змінної с до комірки з адресою 102
01	103	000	000	Увести значення змінної і до комірки з адресою 103
03	101	102	106	Додати значення змінних *Ь* і *с,* результат помістити в комірку з адресою 106, яка відповідає змінній *г*

коп	А1	А2	А3	Примітка
05	106	103	100	Помножити значення змінних *г* і *сі,* помістити результат у комірку з адресою 100 (у змінну *а)*
02	100	000	000	Вивести значення змінної *а* з комірки, що має адресу 100
00	000	000	000	Кінець обчислень (адреси операндів та результату не потрібні)

Нехай програма розміщена в пам'яті так, що адресою її першої команди є комірка 500. Ця адреса запам'ятовується в лічильнику команд (так звана *?гускова адреса),* і включається режим автоматичного виконання програми, що передбачає таку послідовність дій.

1. Перша команда програми, адреса якої визначається лічильником команд, передається в регістр команд процесора.
2. Арифметико-логічний пристрій процесора дешифрує код операції.
3. Числа, що зберігаються в комірках, які визначаються адресами команди, передаються в АЛП як операнди.
4. Виконується операція, і результат заноситься в комірку пам'яті з адресою, зазначеною у команді.
5. Вміст лічильника команд модифікується шляхом додавання до нього довжини поточної команди, і для наступної команди повторюються дії, що зазначені в пунктах 1-5.
6. Обчислення завершується, коли поточна команда містить код операції, яка припиняє обчислення.

Інформацію, призначену для обробки комп'ютером, необхідно закодувати у відповідному форматі. У результаті кодування будь-яке число, символ чи команда перетворюються у рядок двійкових цифр. Для зображення кожного типу даних передбачено певний формат.

1.4. Інформація в пам'яті комп'ютера

Символьна, графічна, числова та будь-яка інша інформація зображується в пам'яті комп'ютера у вигляді послідовностей одиничних та нульових бітів. Такі послідовності можна розглядати як двійкові, вісімкові або шістнадцяткові числа і виконувати над ними арифметичні операції у відповідних системах числення.

1.4.1. Позиційні системи числення

Використання двійкової системи числення набагато спрощує апаратну реалізацію пристроїв комп'ютера. Але двійковий запис числа приблизно втричі довший за його десятковий еквівалент, і тому людині працювати з ним незручно. У програмуванні для скороченого запису двійкових чисел використовують *шістнадцяткову систему числення.* Перші десять цифр шістнадцяткової системи збігаються

з десятьма арабськими цифрами. До них додаються шість латинських літер A, B, C, B, E, P, котрі позначають десяткові числа 10, 11, 12, 13, 14, 15.

Усі згадані системи числення є *позиційними*, оскільки вага кожної цифри залежить від її позиції в записі числа. Наприклад, цифра 3 у числі 31 позначає десятки, тобто має вагу 10^i, а в числі 13 позначає одиниці, тобто її вага становить 10°. Кількість різних символів, що використовуються для запису чисел, називається *основою системи числення.* Позиційна система числення з основою *N* має А цифр C_0, C_b ..., Сдм, шо позначають натуральні числа від 0 до *N* - 1.

Число *N* у А-ковій системі позначається дворозрядним записом CiCo. Так, числа 10 у десятковій, 2 у двійковій та 16 у шістнадцятковій системах записуються однаково: 10. Число A^2 позначається вже трьома цифрами: CiC_0C_0 тощо. Аналогічно, т-розрядні дроби (ти > 0) у А-ковій системі мають вигляд 0
де нулем позначена ціла частина числа, вага цифри *x_i* дорівнює А~', *i* = 1, 2,..., *m*. Так, у десятковому числі 0,1234 цифра 3 має вагу 10~'\

Остання цифра справа у записі цілого числа (або остання цифра перед комою у дробовому числі) називається *наймолодшою*, перша цифра ліворуч називається *найстаршою.* Основу системи числення вказують праворуч від числа у нижньому індексі, наприклад: 123_{10}, 1001_2; якщо це позначення основи опущене, число вважається десятковим. Значення У числа, що записане у Лаковій системі числення, дорівнює значенню такого полінома:

Тут *Xi* - значення цифри в г-му розряді числа; $A^{r<}$- вага г-го розряду числа; *i* - -1, -2, ..., -7?г — розряди дробової частини числа; *i* = 0, 1,.... *n* — розряди цілої частини числа.

Зауважимо, що скінченний Арковий дріб дорівнюватиме скінченному десятковому дробу лише тоді, коли число А"не матиме інших дільників, окрім 5_{10} та 2_{10}. Інакше скінченний А'-ковий дріб буде зображений у десятковій системі у вигляді нескінченного періодичного дробу.

Приклад 1.2

$512_{10} = 5x10^2 + 1x10^i + 2x10°;$

$(512,346)_{50} = 5xЮ^2 + 1x10^i + 2x10° + 3xЮ"^1 + 4x10~^2 + 6x10"^3$;

$(10\ 011)_2 = 1x2^4 + 0x2^3 + 0x2^2 + 1x2^i + 1x2° = 19_{1(}$;

$(10,011)2 = 1x2^i + 0x2° + 0x2^{-1} + 1x2^{\ 2} + 1x2"^3 = 2,375_{10}$;

$(1BC)_{16} = Ixiб^2 + 11x16 + 12 = 444_{10}$;

$(1B,C)_{16} = 1x16^i + 11x16° + 12XI6"^1 = 27,75_{10}$;

$(12,2)_3 = 1x3^i + 2x3° + 2x3^{-1} = 5,(6)_{10}$ (число зображується нескінченним періодичним десятковим дробом).

Оскільки в комп'ютері числа обробляються у двійковій системі числення, а для дюдини звичним є десятковий запис чисел, то постає потреба у переведенні числа з однієї системи числення до іншої.

Переведення натуральних чисел з однієї системи числення до іншої

Розглянемо задачу переведення числа *P* з М-кової системи числення до Аркової.

Нехай число *P* у А-ковій системі числення має запис $P = (x_к x/_{i\_1}..x_i x_0)_№$ який містить цифри X; у невідомій кількості к + і. Записати це число можна так:

$$P = x_к x!^{\wedge} + x \wedge x A \wedge^{1} +...+ x\,\mathbf{ixA} + x_0 = ((.x*xA + x_{кл})xA +...+ x_4)xA + x_0.$$

Звідси видно, що значенням наймолодшої цифри x_0 є остача від ділення числа *P* на основу А (усі операції здійснюються над М-ковими числами). Значенням другої справа цифри x_x буде остача від ділення частки, яку отримано на попередній ітерації, на основу *N*. Продовжуючи ці міркування, отримаємо циклічну процедуру, кожна ітерація якої полягатиме у знаходженні частки та остачі від ділення деякого числа () на основу системи числення. При цьому отримана на *i*-й :терації частка стає самим числом () на (г+1)-й ітерації. Якщо частка менша від А, :5числення завершують. Остання частка є старшою цифрою числа, інші цифри : аписуються в порядку, зворотному до порядку отримання остач, тобто перша ос- : іча дає наймолодшу цифру.

Приклад 1.3

Нехай потрібно перевести число 25_{10} з десяткової системи числення до двійкової. _тя розв'язання цієї задачі застосуємо щойно описану схему обчислень. Остачу з ід ділення будемо записувати в дужках після частки,

15:2 = 12(1);
— : 2 = 6(0);
5:2 = 3(0);
3:2 = 1(1),

Остання частка менша двох. Вона є старшою цифрою двійкового числа, до кого треба дописати остачі у порядку, зворотному до порядку їх отримання. Результатом є число 11001_2.

З математичної точки зору переведення чисел з будь-якої М-кової позиційної : ястеми числення до будь-якої А-кової здійснюється за тією процедурою, що її :уло розглянуто вище. Проте в цій процедурі операції знаходження частки та : ітачі від ділення здійснюються над числами у М-ковій системі, а для людини і зичними є лише операції над десятковими числами. Тому, як зазначалося вище, десяткове значення М-кового числа Рзручніше обчислювати, користуючись його : ображенням у вигляді поліному $P = x\hat{}M* + x\wedge xM*"^{1} +... + x_4xM + x_0$.

Наприклад;

$5AP_{16}=5_{16}x\ 16^2 + A.йx\ 16^i + P_{16}x\ 16°= 5\ x\ 16^2+ 10\ x\ 16^i + 15\ x\ 16° =1455;$

і $10017 = 1x2^4 + ix2^3 + 0x2^2 + 0x2^i + 1x2° = 25.$

Переведення дробових чисел з однієї системи числення до іншої

Нехай задане М-кове дробове чисто *V,* (0<Уй1). Припустимо, що у ЛГ-ковій системі числення число У має запис *У*= (0д:_і.г-_2...)лг. тобто

$$У = АгЬ<(x\_, + iУ-^1x(x_{-2} +...))•$$

Тут цифри невідомі. Помножимо обидві частини рівності на А:

$$1/XA=X„i + Л^{r1}x(x„2 +...).$$

Значення цифри *x_,* отримаємо, взявши цілу частину добутку УхЛГ, тобто *x.*\ = = [.УхЛ^І а /у' ^{1}х(х_г) = {УхА^1}, де і УхА] та {УхА} позначають цілу та дробову частини У<А. Помножимо дробову частину {УхА} на А і знову візьмемо цілу частину В результаті отримаємо значення цифри x_{-2} і т. д. Якщо на деякій ітерації значення {УхА^7} стане рівним 0, процес можна припинити. Але, можливо, значення {УхА} ніколи нулю не дорівнюватиме. В такому разі запис точного значення У у А-ковій системі буде нескінченним, а кожна додатково отримана цифра уточнюватиме наближене значення У.

Отже, для переведення дробу з М-кової системи числення до А'-кової потрібно послідовно множити дробову частину на основу системи числення А. Цілі частини добутків, отриманих у результаті виконання послідовності операцій множення, є цифрами дробу в Лаковій системі числення.

У разі, коли У задане у вигляді скінченного десяткового дробу і число А має серед своїх простих дільників як 2, так і 5, зображення числа У в ЛГ-ковій системі числення буде скінченним. Інакше в А-ковій системі числення десятковий дріб У може зображуватися нескінченним періодичним дробом.

Приклад 1.4

Нехай потрібно перевести десятковий дріб 0,75 до двійкової системи. Послідовність операцій множення така: 0,75х2 = 1,5; ціла частина 11,5] = 1, дробова частина {1,5} = 0,5; 0,5х2 = 1, дробова частина {1} = 0. Усі подальші цифри будуть нулями, тому $0,11_2$ є скінченним двійковим зображенням дробу $0,75_{10}$.

У процесі переведення десяткового Дробу $0,26_{10}$ у двійкову систему виконується така послідовність операцій множення* 10,26х2] = [0,52] = 0; ІД52х2_| = [і,04] = =1; 10,04х2] - [0,08 і - 0:$^!$..0,08х2. і = [Діб] = 0;! .0,16х2] = 10,32]= 0; |_0,32х2] =[0,64] = = 0, [0,64х2] -;.1,2Й= 1 і очне двійкове зображення десяткового дробу 0,26 є нескінченним періодичним, а $0,010\ 000\ 1_2$ — його наближення.

Переведення десяткового дробу 0,8 у шістнадцяткову систему числення здійснюється так: 0,8х1.6 = 12,8 з цілою частиною |_12,8] = 12 і дробовою частиною {12,8} = 0,8. Десяткове число 12 замінюємо на шістнадцяткове значення C_{16}. Усі подальші шістнадцяткові цифри будуть також дорівнювати *С.* Отже, маємо періоди<ший іріб $0,\{C\)_{.6}$.

Зв'язок двійкової та шістнадцяткової систем

Розглянемо простий спосіб переведення шістнадцяткового числа у двійкове та навпаки. Оскільки $16 = 2^4 = 10000_2$, то одна шістнадцяткова цифра використовується для зображення чотирьох бітів:

0 - 0000, 1 - 0001, 2 - 0010, 3 - 0011, 4 - 0100, 5 - 0101, 6 - 0110, 7 - 0111,
З - 1000,9- 1001, А - 1010, В - 1011, С- 1100,0- 1101, Е- 1110, Р- 1111.

Отже, для переведення двійкового числа у шістнадцяткове достатньо у двійковому записі, починаючи з наймолодшої цифри, замінити кожні чотири біти відповідними шістнадцятковими цифрами (якщо найстарша четвірка розрядів неповна, до неї дописуються незначущі нулі). Навпаки, шістнадцятковий запис дереводиться в двійковий заміною цифр відповідними четвірками бітів. Четвірка бітів, що відповідає найстаршій шістнадцятковій цифрі, може мати незначущі нулі у старших розрядах.

При переведенні дробових чисел четвірки розрядів слід виділяти, починаючи віл найближчого до коми розряду як у цілій, так і у дробовій частині числа. При дьому до неповних четвірок дописуються нулі (йдеться про наймолодшу четвірку розрядів у дробовій частині та найстаршу четвірку у цілій частині числа).

Приклад 1.5_

Перевести в шістнадцяткову систему числення двійкове число 1111010. Спочат-•у число, починаючи від молодшої цифри, розбивають на групи: 1010 та 111. Ітарша група доповнюється до тетради 0111. Цій тетраді відповідає шістнадцяткова цифра 7, а тетраді 1010 — цифра А. Отже, результатом переведення є 7А.

Тепер зобразимо у двійковому вигляді шістнадцяткове число А7. ІПістнаддяткова цифра А зображує двійкове число 1010, а шістнадцяткова цифра 7 - двійкове число 111. Доповнивши двійковий запис числа 7 нулем у старшому роздяді, отримаємо $А7_{16} = 10100111_2$.

Арифметичні операції в різних системах числення

Зсновними арифметичними операціями у системі команд будь-якого комп'ютера -: додавання, віднімання, множення та ділення. Правила виконання цих операцій у десятковій системі числення загальновідомі. Вони застосовуються і при ви->: знанні аналогічних дій в інших позиційних системах числення.

Розглянемо операцію додавання двійкових чисел. У десятковій системі 9 + 1 = = 10, а у двійковій 1 + 1 = 10. Таким чином, при додаванні у стовпчик двох одиничних розрядів отримаємо нульовий розряд суми і одиницю перенесення у наітупний, старший розряд. Додамо у стовпчик числа 11_2 і 110_2, вказуючи розряди деоенесень у дужках.

```
 (110)
  11
 110
1001
```

Додавання у системі числення з довільною основою Р виконується аналогічно тому, як це робиться у десятковій та двійковій системах. При додаванні двох однорозрядних Р-кових чисел $a\backslash$ і a_2 обчислюється сума $5 = ai + a_2$, і результатом є (5 <ііу Р)5 тосі P, де в дужках вказано одиницю перенесення у наступний розряд, 5 (ІІУ P — ціла частина, а 5 тосі P — остача від ділення 5 на P. При додаванні багаторозрядних Р-кових чисел, починаючи від молодших розрядів, обчислюються суми розрядів і одиниць перенесення від попереднього розряду. Остача від ділення на Р залишається у відповідному розряді результату, а частка переноситься в наступний розряд.

Приклад 1.6

У цьому прикладі обчислимо суму двох десяткових чисел 548ю і 54_{10}, а також їх вісімкових та шістнадцяткових еквівалентів - 1044_8 і 66_8, 224_{16} і 36_{16} (у дужках зверху над значеннями доданків вказуються одиниці перенесення у відповідні розряди).

(110)	(110)	
548_{10}	Ю44$_8$	+ 224_{16}
54_{10}	66_8	36_{16}
602_{1o}	1132_8	$25A_{16}$

Віднімання в будь-якій системі числення виконується за тими самими правилами, що й у десятковій.

Приклад 1.7

Віднімемо одиницю від чисел 100_2, 100_8 і 100_{16}:

100_2	Ю0$_8$	100_{16}
I2	l»	1,6
11_2	77_8	pp,6

Операцію множення у довільній системі числення можна виконати за допомогою методу, який подібний до методу множення у стовпчик. Обчислення добутку двох п-розрядних чисел без знака зводиться до формування часткових добутків для кожної цифри множника і їх подальшого додавання. Коли обидва операнди є двійковими числами, процедура обчислення часткових добутків значно спрощується: якщо цифра множника дорівнює нулю, то і частковий добуток теж становить нуль, якщо одиниці, частковий добуток дорівнює значенню множеного. Перед додаванням кожний частковий добуток необхідно зсунути на один розряд вліво відносно попереднього добутку. Таким чином, результатом множення двох п-розрядних двійкових чисел може бути число, яке має 2п розрядів.

Приклад 1.8

Обчислимо добуток двох чисел у десятковій, вісімковій і шістнадцятковій системах числення. Співмножниками будуть числа $29_{10} = 35§ = Шіб > 19ю = 23_8 = 13_{16}$.

$29,_0$		35,ч		Ю.6
19_{10}	x	23_8	x	$із_{16}$
261,0		127_8	+	$57,_6$
29,о		72_8		Юіб
551ю		1047_8		$227,_6$

1.4.2, Зображення чисел у комп'ютері

Комп'ютери всіх типів мають велику кількість команд для виконання арифметичних операцій над числовими даними. Для того щоб зрозуміти, як комп'ютер виконує ці команди, необхідно знати, як числа зберігаються в пам'яті і як вони додаються та віднімаються. Зрозуміло, що в розрахунках використовуються як додатні, так і від'ємні числа, і нам потрібно їх певним чином зобразити у пам'яті комп'ютера.

У сучасних комп'ютерах здебільшого застосовуються два формата зображення чисел: числа з фіксованою комою і числа з плаваючою комою (іїхесі роті, їїоапп§ роіпі). Перша з них дістала назву природної (або натуральної), друга — нормальної (експоненціальної, логарифмічної або так званого наукового запису).

Числа з фіксованою комою

Число з фіксованою комою - це формат зображення числа з незмінним розташуванням коми, що відокремлює цілу частину числа від дробової. Числа у такому форматі записуються у вигляді $X= +a_{n\_i}..a_i\ a_0$ a_і $a_{\_2}...a_{\_r}$ (рис. 1.3), де для запису цілої частини числа відводиться *п* розрядів, а для дробової частини числа — *г* розрядів.

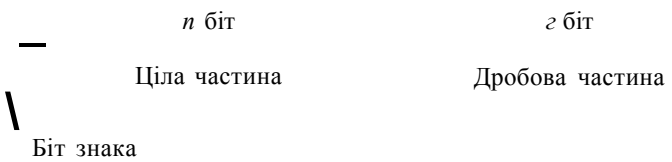


Рис. 1.3. Формат запису чисел із фіксованою комою

Розряд коду числа, в якому вказується знак, називається знаковим, а розряди, де знаходяться значущі цифри, називаються цифровими розрядами коду. Знаковий розряд дорівнює 0 для додатних чисел, та 1 — для від'ємних. Положення коми зідносно розрядів числа фіксується й у процесі обчислень не змінюється. В самому коді числа кома фізично ніяк не вказується, вона лише «мається на увазі».

У комп'ютері числа з фіксованою комою є одним із базових різновидів числових даних. Припустивши, що число не має дробових розрядів, відразу одержимо множину цілих чисел. Отже, числа з фіксованою комою можна умовно поділити на цілі числа і числа з дробовою частиною. Для роботи з цілими (знаковими і беззнаковими) числами, розмір яких у пам'яті становить 1, 2,4 і 8 байт, процесор має спеціальні команди. Для процесорів IiHeI характерним є те, що в них не використовуються числа з фіксованою комою, якщо не вказувати даних, які зображують цілі числа.

Якщо число має цілу та дробову частини (є мішаним), то арифметичні дії над ним виконуються так, нібито це число є цілим (хоч насправді воно не є таким). Для спрощення операцій над такими числами положення коми фіксується або перед старшим цифровим розрядом, або після молодшого. У першому випадку можуть бути зображені тільки правильні дроби (за модулем менші одиниці), у другому — тільки цілі числа. Останній формат найбільш поширений, тому надалі поняття «фіксована кома» зв'язуватимемо з цілими числами, а операції з числами у форматі з фіксованою комою характеризуватимемо як операції над цілими числами (зі знаком і без знака).

Використання чисел у форматі з фіксованою комою значно спрощує апаратну реалізацію арифметико-логічного пристрою комп'ютера і зменшує час виконання машинних команд.

Для зображення додатних та від'ємних цілих чисел у комп'ютері застосовуються прямий, обернений та додатковий коди. Додатні числа у прямому, оберненому та додатковому кодах записуються однаково, а саме двійковим кодом числа з цифрою 0 у знаковому розряді, а від'ємні — по-різному.

Прямий код від'ємного числа відрізняється від прямого коду додатного числа тим, * о значення знакового розряду (старшого біта числа) дорівнює не 0, а 1. Наприклад, прямим кодом числа 5 є 0101, а числа 127 — 01111111. Відповідно, прямий код числа -5 становить 1101, а код числа -127 записується так: 11111111.

Обернений код (чи доповнення до одиниці) від'ємного числа можна отримати шляхом доповнення кожного розряду відповідного додатного значення до одиниці, тобто у всіх розрядах, у тому числі знаковому, нулі потрібно замінити одиницями, а одиниці - нулями. Ця операція еквівалентна відніманню цього числа від 2^n-1 (наприклад, від 1111 для чотирирозрядних чисел). Таким чином, обернений код числа -5 записується як 1010, а код числа -127 - як 10000000.

Додатковий код від'ємного числа можна отримати додаванням одиниці до молодшого розряду оберненого коду або відніманням модуля числа від 2^n. Наприклад, додатковий код числа -5 записується як 1011, а код числа -127 - як 10000001.

Порівнюючи всі три коди з погляду ефективності виконання арифметичних операцій комп'ютером, слід зазначити, що прямий код найменше підходить для виконання арифметичних операцій, обернений код є більш придатним, а найефективнішим вважається додатковий код.

Арифметичні операції над цілими числами

У пам'яті комп'ютера операнди, як правило, зберігаються у прямому або додатковому кодах. Зберігати числа в оберненому коді недоцільно, тому що такий код

числа можна дуже легко отримати з прямого коду простою інверсією. Використання прямого коду числа дає переваги під час виконання деяких операцій (наприклад, множення) над операндами, що зображені у форматі з плаваючою комою. Ця обставина і є визначальним фактором при виборі способу збереження інформації у пам'яті комп'ютера.

При додаванні двох «-розрядних двійкових чисел (один біт знака та *n* -1 бітів значущих цифр) слід керуватися такими правилами.

1. Для того щоб додати два числа, необхідно додати їх и-бітове зображення, не враховуючи біт перенесення, який формується у старшому розряді. Сумою буде значення у додатковому коді, якщо результат належить діапазону від -2 до 2^{n-1} - 1.
2. Для віднімання числа X від числа 7 необхідно спочатку обчислити додатковий код числа У, а потім додати його до числа X, враховуючи перше правило. Результатом буде значення у додатковому коді, за умови, що воно належить діапазону від -2^{n-1} до 1.

При додаванні двох и-розрядних двійкових чисел (один біт знака та *n*-1 бітів значущих цифр) із використанням прямого та додаткового кодів можливий результат, кількість значущих цифр у якому дорівнюватиме *n,* тобто результат, що не належатиме діапазону допустимих значень. Таку ситуацію називають *переповненням* розрядної сітки. Ознакою переповнення є перенесення в знаковий розряд результату за відсутності перенесення зі знакового розряду (*додатне переповнення)* або наявність перенесення зі знакового розряду за відсутності перенесення з знаковий розряд (*від'ємне переповнення).* Якщо є перенесення як в знаковий розряд результату, так і зі знакового розряду або якщо ці перенесення відсутні, то переповнення не відбувається. При додатному переповненні результат операції від'ємний, при від'ємному — додатний.

Слід зазначити, що переповнення може виникнути лише при додаванні чисел і однаковими знаками. Додавання чисел із різними знаками не призводить до переповнення.

Приклад 1.9_

Наведемо приклад додатного переповнення. В області пам'яті обсягом 1 байт зберігається число +65ю, прямим кодом якого є 0100 0001 (знаковий розряд містить 0). Якщо додати до цього числа таке саме, утвориться від'ємне число 1000 0010, оскільки відбудеться перенесення одиниці в знаковий розряд (знаковий біт результату міститиме 1). Десятковим еквівалентом числа, що утвориться, буде значення -2_{10}.

```
  0100 0001
+ 0100 0001
  1000 0010
```

Далі наведемо приклад від'ємного переповнення. В області пам'яті обсягом 1 байт зберігається число -65_{10}, додатковий код якого — 1011 1111 (знаковий розряд містить 1). Якщо додати до цього числа таке саме, то утвориться додатне

число 0111 1110, оскільки відбудеться перенесення одиниці зі знакового розряду (знаковий біт результату міститиме 0). Десятковим еквівалентом двійкового числа, що утвориться, буде число $+126_{10}$.

```
  1011 1111
^ 1011 1111

  0111 1110
```

Виявити переповнення розрядної сітки буде простіше за умови використання *модифікованого додаткового коду,* коли для збереження знака числа відводиться не один, а два розряди. Ці розряди беруть участь в арифметичній операції нарівні з цифровими. За відсутності переповнення обидва знакові розряди містять однакові значення. Розбіжність у значеннях цих розрядів є ознакою того, що виникло переповнення.

Множення, порівняно з додаванням та відніманням, є більш складною операцією з точки зору як програмної, так і апаратної реалізації. У більшості комп'ютерів операція множення виконується як послідовність операцій додавання та зсуву. Для реалізації цієї операції у комп'ютерах застосовуються кілька різних алгоритмів, але кожен із них передбачає наявність в АЛП регістрів множника та множеного, а також спеціального регістру, який називається *накопичувальним а/матором.* До початку операції множення цей регістр містить число 0. Під час операції у ньому зберігаються результати проміжних обчислень, а по завершенні — кінцевий результат.

Приклад 1.10_

Приклад множення двох додатних чисел наведено нижче.

```
     1101
  x  |0!1
     иоГ
    1101
  1101
  ТооошГ
```

Перевіримо:

$1101_2=13$,

10112=11,

13x11=143,

$10001111_2=2^7+2^3+2^2+2^1+2^0=128+8+4+2+1=143$

Ділення є дещо складнішою, ніж множення, операцією, але її реалізація в комп'ютері ґрунтується на тих самих принципах. Основою є загальновідомий спосіб ділення за допомогою операцій віднімання чи додавання та зсуву (ділення у стовпчик). Ділення виконується як послідовність віднімань дільника спочатку від ді-

леного, а потім і від часткових залишків, які утворюються у процесі ділення. Ділене зазвичай займає 2п розрядів, тоді як дільник, частка та залишок є п-розрядними.

Числа з плаваючою комою

У прикладних задачах програмістам досить часто доводиться оперувати дуже великими або дуже маленькими дійсними числами, наприклад такими, як маса Сонця, що складає $2x10^{30}$ кг, або маса електрона, яка становить $9x10^{-28}$ г. Записати в пам'ять подібні числа, враховуючи всі значущі цифри, і виконати над ними арифметичні операції, використовуючи арифметику з фіксованою комою, неможливо. У цьому разі для запису чисел використовується формат із плаваючою комою, коли кожне число розбивається на дві групи цифр. Перша група цифр називається мантисою, друга - порядком. Число записується у вигляді добутку.

$$У = \pm Mx \ 5^{\pm p}.$$

Тут *У* - значення дійсного числа; *М* - мантиса числа; *5* - основа системи числення; *р* — порядок числа.

Мантиса (дріб зі знаком) і порядок (ціле число зі знаком) зображуються в системі числення з основою Знак числа збігається зі знаком мантиси. *Порядок р* є додатним або від'ємним цілим числом і визначає положення коми в числі *У*. Таким чином, у мантисі зберігаються значущі цифри числа, а порядок визначає його величину.

Дійсні числа записуються в 4, 6, 8 або 10 байтах. Для того щоб в комірці оперативної пам'яті не зберігати знак порядку, замість нього використовується характеристика. Вона утворюється додаванням до порядку певного цілого числа. Це число добирається так, щоб характеристика завжди була додатною. Характеристику іноді називають *зсуненим порядком.* При цьому формат зображення дійсних чисел є таким, як показано на рис. 1.4.

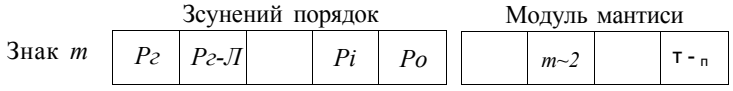


Рис. 1.4. Формат чисел із плаваючою комою зі зсуненим порядком

Для збільшення кількості значущих цифр у зображенні дійсного числа і запобігання переповненню при виконанні арифметичних операцій мантису нормадізують. *Нормалізація* означає, що значення мантиси має знаходитися в діапазоні зід $У^1$ до 1, де 5 - основа системи числення. У двійковій системі числення *5-2* - мантиса набуває значення від $2"^1$ до 1. Це означає, що мантиса будь-якого числа, зображеного у форматі з плаваючою комою, має починатися з одиниці у двійковій системі числення. Наведений метод нормалізації є класичним, при якому результат нормалізації зображується у вигляді правильного дробу, тобто з одиницею після коми і нулем у цілій частині числа (розглядається двійкова система). Є різні алгоритми нормалізації мантиси. В IBM-сумісних комп'ютерах старший біт нормалізованої мантиси розташований зліва від коми. В оперативній пам'яті

цей біт не зберігається, тобто він є прихованим, а його вага дорівнює одиниці, тому мантиса належить інтервалу і < М< 2, Нормалізована мантиса додатних і від'ємних чисел зображується у прямому коді.

Діапазон чисел з плаваючою комою залежить від кількості розрядів, виділених для зображення порядку і мантиси, а також від основи системи числення. Слід зазначити, що розташування та довжини полів у зображеннях дійсних чисел визначаються типом комп'ютера та мовою програмування.

Визначимо діапазон дійсних чисел, що зображуються чотирма байтами (32 двійковими розрядами). У нормалізованій мантисі перша значуща цифра записана зліва від коми, а справа розташовуються 23 розряди (рис. 1.5). Тому максимальне значення мантиси $M_{max} = 1{,}111..11 = 1+1/2+1/4+1/8+...= 2$, а її мінімальне значення $M^{\wedge} = 1.000..00 = 1$ для додатних чисел і $M_{min} = -1$ та $M_{min} = -2$ для від'ємних чисел. Максимальне значення порядку числа визначається як $P_{max} = 2^8 - 2 = 254_{10} = 11111110_2$, а мінімальне значення порядку - як $P_{min} = 00000001 = 1$. Після цього можна визначити діапазон зображення додатних дійсних чисел: від $Д^{\wedge} = M_{min} \times 2^{(1-127)} \approx 1{,}17 \times Ю^{-38}$ до $£_{max} = M_{max} \times 2^{254-127} \approx 3{,}4 \times 10^{38}$.

Приклад 1.11

Зобразимо десяткове число $-15{,}375_{10}$ у форматі з плаваючою комою. Спочатку переведемо десяткове число в двійкову систему числення і отримаємо його двійковий еквівалент, тобто $-1111{,}011_2$,. Після цього нормалізуємо число і отримаємо значення $-1{,}111011 \times 2^3$, де порядок числа дорівнює трьом. Оскільки число від'ємне, то знаковий біт міститиме одиницю. Обчислюючи характеристику числа, отримаємо значення $2^7 - 1 + 3 = 130_{10}$. Переведення значення характеристики в двійкову систему дає результат $1000\ 0010_2$. Одиниця цілої частини нормалізованого числа відкидається, тому мантиса дійсного числа має вигляд 111011_2. Тепер можна записати машинне зображення дійсного числа, використавши для цього чотири байти:

11000001 01110110 00000000 00000000.

Точність обчислень при використанні чисел із плаваючою комою визначається кількістю розрядів мантиси, тобто числом достовірних десяткових цифр. Оскільки 2^{23} приблизно дорівнює 10^7, при наявності 23 двійкових розрядів мантиси достовірними є тільки 6-7 значущих десяткових знаків.

Комп'ютери, які серійно випускаються різними фірмами, використовують різні формати зображення чисел. В першу чергу це стосується чисел із плаваючою комою. Щоб спростити використання програм, розроблених для різних платформ, було запропоновано стандарт IEEE 754, який регламентує формат запису чисел із плаваючою комою. Нині цей стандарт широко використовується, і практично всі розробники комп'ютерів дотримуються його вимог. Цей стандарт визначає два базових формати (32- та 64-бітовий), з 8- та 11-розрядними порядками відповідно (рис. 1.5).

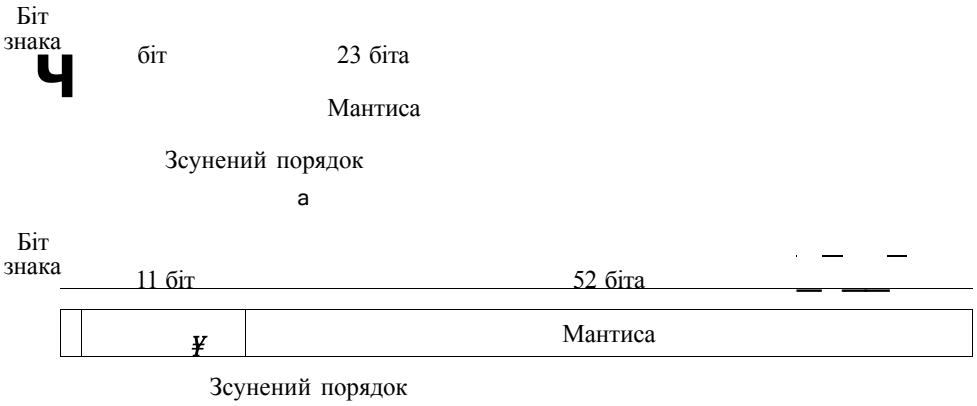


Рис. 1.5. Основні формати чисел з плаваючою комою в стандарті IEEE 754: 32-бітовий формат (а); 64-бітовий формат *(б)*

Крім двох основних, у стандарті IEEE 754 запропоновано також два розширених формати, одинарний та подвійний, в яких зарезервовано додаткові біти для порядку та мантиси.

1.5. Типи комп'ютерів

Різнотиповість комп'ютерів обумовлюється передусім різнотиповістю обчислень і задач, для виконання яких вони призначені. З часом виникають все нові класи задач, що, у свою чергу, приводить до появи комп'ютерів нових типів. Тому наведена нижче класифікація містить лише найбільш поширені типи комп'ютерів:

- суперкомп'ютери;
- мейнфрейми;
- сервери різних типів;
- персональні комп'ютери та робочі станції.

Суперкомп'ютери — спеціальний тип комп'ютерів, що створюється для розв'язування надзвичайно складних обчислювальних задач (підготовки прогнозів погоди, моделювання складних процесів та явищ природи, оброблення великих обсягів інформації). Найпотужніший у світі суперкомп'ютер (ЕатіЬ Зітіаїог), за даними 2003 року, встановлено в Центрі моделювання Землі в Йокогамі (Японія). Він призначений для моделювання основних властивостей складових кліматичної системи Землі: атмосфери, океану, кріосфери, поверхні суші і біосфери, а також зовнішніх і внутрішніх факторів у системі, яка визначає глобальний клімат і його зміни. Суперкомп'ютери — це багатопроцесорні або багатомашинні комплекси продуктивністю понад сотні мільярдів операцій за секунду. Основний принцип роботи всіх суперкомп'ютерів — принцип паралелізму. Суть його полягає в тому, що комп'ютер здатний виконувати одночасно (паралельно) велику кількість операцій.

Однією з провідних компаній світу по виробництву суперкомп'ютерів багато років вважалась компанія Сгау КезеагсЬ. Її засновник, людина-легенда Сеймур Крей, уже в середині 70-х років побудував комп'ютер Сгау-1, що вражав світ своєю швидкодією: десятки і навіть сотні мільйонів арифметичних операцій за секунду. Сьогодні на ринку суперкомп'ютерів домінують більш сучасні системи компаній N£0, HP, IBM.

Мейнфрейм — це синонім поняття «велика універсальна електронно-обчислювальна машина». Мейнфрейми і на сьогодні залишаються найбільш потужними (якщо не враховувати суперкомп'ютери) обчислювальними системами загального призначення, які можуть експлуатуватися у безперервному режимі. Зазвичай мейнфрейми - це багатопроцесорні системи, що містять один або кілька центральних і периферійних процесорів із спільною пам'яттю, зв'язаних між собою високошвидкісними магістралями передачі даних. При цьому основне навантаження щодо обчислень покладено на центральні процесори, а периферійні процесори забезпечують роботу з різноманітними периферійними пристроями.

Основними постачальниками мейнфреймів є відомі комп'ютерні компанії АшсіаЫ, ІСЬ, Зіешепз Мхсіогі і деякі інші, але провідна роль у цій галузі, безумовно, належить компанії IBM. Саме архітектура випущеної в 1964 році системи IBM/360 і її наступних поколінь стала зразком обчислювальних систем цього типу. В СРСР протягом багатьох років випускалися машини серії ЄС ЕОМ (Єдина серія електронно-обчислювальних машин), що були вітчизняним аналогом цієї системи.

Комп'ютер, що працює в локальній або глобальній мережі, може спеціалізуватися на обслуговуванні інших комп'ютерів. Такий комп'ютер називається *сервером* (від англ. 8егуе - обслуговувати, керувати). Є кілька типів серверів, орієнтованих на різні сфери застосування: файловий сервер, сервер бази даних, сервер друку, обчислювальний сервер, сервер програмних застосувань. Тип сервера визначається видом ресурсу, яким він керує (файлова система, база даних, принтери, процесори або пакети прикладних програм).

Існує також класифікація серверів на основі масштабу мережі, в якій вони використовуються: сервер робочої групи, сервер відділу або сервер масштабу підприємства (його ще називають корпоративним сервером). Загалом, ці класифікації досить умовні. Наприклад, розмір групи може змінюватися в діапазоні від кількох людей до декількох сотень, а сервер відділу може обслуговувати від 20 до 150 користувачів. Очевидно, що вимоги до складу устаткування і програмного забезпечення сервера, його надійності і продуктивності дуже варіюються залежно від числа користувачів і характеру розв'язуваних ними задач.

Незважаючи на велике різноманіття типів і моделей обчислювальних систем, у більшості користувачів поняття «комп'ютер» асоціюється в першу чергу з персональним комп'ютером. Перший персональний комп'ютер з'явився у 1975 році. І відразу стало зрозуміло, що невисока ціна і досить потужні обчислювальні можливості комп'ютерів цього класу сприятимуть їхньому швидкому поширенню.

Персональні комп'ютери зробили революцію у професійній діяльності мільйонів людей, вплинули на всі сфери розвитку суспільства. Комп'ютери цього типу стали незамінним інструментом у роботі інженерів і вчених. Особливо значною

їхня роль є при проведенні наукових експериментів, що вимагають складних і тривалих обчислень.

Сучасний світ наповнений не лише звичними персональними комп'ютерами, а й комп'ютерами-невидимками — мікропроцесорами, що є комп'ютерами у мініатюрі. Крім блоку обробки даних такий пристрій містить блок керування і навіть пам'ять. Це означає, що мікропроцесор здатний автономно виконувати всі необхідні дії з інформацією. Масове поширення мікропроцесори одержали у сучасній техніці, якою можна керувати за допомогою обмеженої послідовності команд. Наприклад, керування сучасним двигуном (забезпечення економних витрат палива, обмеження максимальної швидкості руху, контроль за справністю тощо) немислиме без використання мікропроцесорів. Ще однією сферою їхнього використання є побутова техніка — застосування мікропроцесорів додає їй нових споживчих якостей.

1.6. Програмне забезпечення

Під програмним забезпеченням розуміють сукупність усіх програм і службових даних, призначених для керування комп'ютером. Деякі програми є вбудованими в апаратні компоненти комп'ютера, однак для забезпечення більшої гнучкості їх зазвичай записують на жорсткий диск, компакт-диск або інші зовнішні носії даних. У цьому разі їх необхідно щораз заново завантажувати в оперативну пам'ять при запуску комп'ютера чи перед виконанням конкретної програми. Програмне забезпечення складається з файлів програм, що керують роботою комп'ютера.

Файл — це послідовність літерно-цифрових символів або двійкових даних; до нього можна звертатися за вказаним іменем. У файлах зберігаються програми та дані. Кожний файл має ім'я, розмір, дату створення та певне місце розташування на диску. Залежно від характеру інформації, яка записується у файл, він може мати те чи інше розширення імені, що задається трьома латинськими літерами. Наприклад, текстові файли можуть позначатись розширенням ЬЛ, графічні файли - розширенням Ьтр, програми — розширенням ехе тощо. Ім'я, розширення імені, дата створення, розмір файла є його атрибутами. Значення атрибутів файла зберігається у каталогах (папках, директоріях).

Каталогом називається група файлів, об'єднаних одним іменем. Слід зазначити, що каталог можна включити до іншого каталогу. Головний каталог диска називається *кореневим каталогом.* Ім'я кореневого каталогу складається з імені диска та символу двокрапки. Місцезнаходження файла на диску визначається маршрутом, який записується у вигляді послідовності імен каталогів, починаючи з кореневого, наприклад: С:\У/іпс!о\л/5\5у5(:ет. Повне ім'я файла, або його специфікація, записується у вигляді маршруту та імені файла: С:\\Л/іпсіоте\5у5І:ет\ІіеІр.ехе.

За своїм призначенням усе програмне забезпечення можна розділити на дві основні категорії:

- системні програми;
- прикладні програми.

Системні програми, або *системне програмне забезпечення,* — це набір програм, призначених для виконання таких функцій:

- отримання й інтерпретація команд користувача;
+ керування процесами збереження файлів на зовнішніх запам'ятовуючих пристроях, а також зчитування інформації із зазначених пристроїв в оперативну пам'ять;
- запуск і керування процесом виконання прикладних програм, таких як текстові редактори, електронні таблиці або ігри, а також програм, створених користувачем;
- керування взаємодією апаратних і програмних ресурсів комп'ютера під час виконання прикладних програм.

Отже, системне програмне забезпечення відповідає за координування всіх операцій, що їх пристрої комп'ютерної системи виконують під час реалізації прикладних програм.

Прикладні програми призначені для розв'язання задач певних класів, наприклад для математичних обчислень, оброблення рядків тексту або відеоінформації. Для розробки прикладних програм використовуються мови програмування, і зокрема C, C++, Вазіс, які дозволяють програмістові вказати дії, що їх має виконати програма.

Ключовим компонентом системного програмного забезпечення є *операційна система* (ОС) — комплекс програм, який використовується для керування взаємодією різних пристроїв комп'ютера при виконанні прикладних програм. Компоненти операційної системи відповідають за надання прикладним програмам ресурсів комп'ютера — оперативної пам'яті і пам'яті на магнітних дисках, пристроїв введення-виведення тощо.

Операційна система, з одного боку, виступає як інтерфейс між апаратними засобами комп'ютера і користувачем, а з іншого, забезпечує ефективне використання останнім ресурсів комп'ютера й організацію виконання прикладних програм. Усі компоненти програмного забезпечення комп'ютера працюють під керуванням операційної системи і жоден з них не має безпосереднього доступу до апаратури. Навіть сам користувач взаємодіє зі своїми програмами через інтерфейс операційної системи.

Для того щоб виконати програму, її потрібно спочатку завантажити до оперативної пам'яті із зовнішнього носія, як правило, з диска. Це робиться під час виконання спеціальної програми, що входить до складу операційної системи, — *завантажувача.* Процес переписування програми із зовнішнього носія в оперативну пам'ять називається *завантаженням.* Як тільки це буде зроблено, програма почне виконуватись (рис. 1.6).

Коли обчислення завершаться, прикладна програма знову направить запит операційній системі. В результаті буде запущено відповідну програму операційної системи, що забезпечить пересилання потрібних даних на пристрій введення-виведення. При виконанні прикладної програми керування періодично передається то їй самій, то програмам операційної системи.

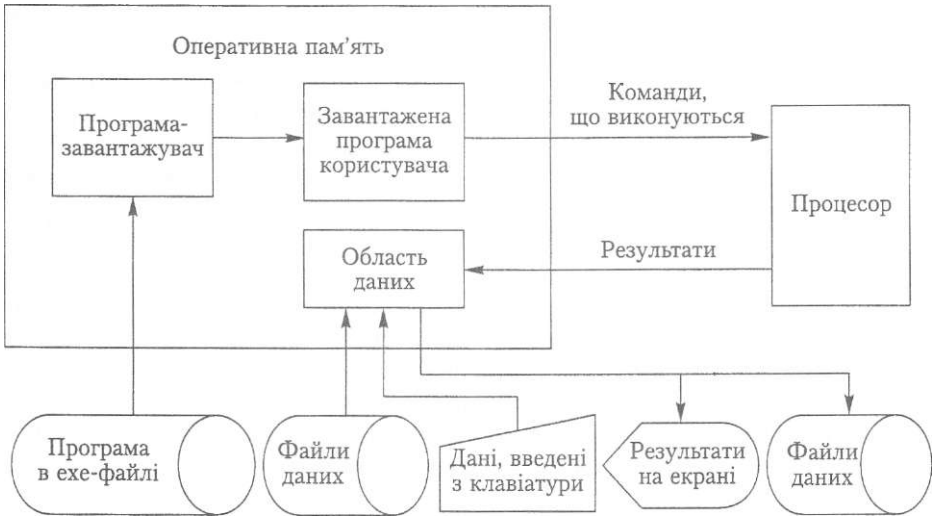


Рис. 1.6. Процес завантаження і виконання програми

1.7. Засоби створення програм

До засобів створення програм належать насамперед мови і системи програмування. Основна функція всіх мов програмування, крім машинної, полягає у тому, щоб надати програмісту засоби абстрагування від характеристик та особливостей апаратного забезпечення, на якому виконуватимуться програми. Системи програмування містять автоматизовані засоби розробки програм.

1.7.1. Класифікація мов програмування

У прикладі 1.1 використовувалися числові коди операцій і числові значення адрес комірок пам'яті. Такий спосіб написання програм називається *програмуванням у числових кодах,* а мова, якою записуються такі програми, — машинною. *Машинна мова* — це «природна мова» певного комп'ютера, яка визначається під час проектування його апаратних засобів. Машинні мови важкі для людського сприйняття. Тому природним виявилося прагнення автоматизувати процес написання програм, для того щоб полегшити працю програміста, частково поклавши його роботу на саму машину. Програмісти почали використовувати більш звичну для людини символьну форму опису обчислень, а перетворення програм у машинні коди здійснювали за допомогою програм трансляції (від англ. їгапзіаїіоп — переклад), які називаються *асемблерами* (від англ. Го аззетЬІе — збирати).

Мови програмування, у яких числове кодування команд було замінено їх символьним зображенням, називалися мовами символічного кодування, а системи програмування - системами символічного кодування (ССК). Нині такі мови перетворилися в досить потужні засоби програмування, названі асемблерами.

Під час написання програм мовами асемблерного типу в ролі засобів програмування використовуються такі абстракції, як змінна та символьне зображення операцій, що дає змогу програмісту позбутися проблем, пов'язаних із формою зображення чисел, кодуванням операцій і розподілом пам'яті. Тепер перераховані вище проблеми має вирішувати програма-транслятор на основі інформації, яку їй передає розроблювач програм. Наступний фрагмент програми мовою асемблера є іншою реалізацією програми з прикладу 1.1.

Приклад 1.12

Припустимо, що комірки оперативної пам'яті з адресами 100, 101, 102 і 103 позначено відповідно як ¥/ОКВА, Ж)ЇШВ, \¥ОКВС, \¥ОЇШВ. Для виконання операцій необхідні регістри процесора, які позначено як АХ і ВХ. Команди «прочитати» й «записати» позначено як МОУ, «додати» і «помножити» — як АББ та МІ)Т. Тоді послідовність команд із прикладу 1.1 матиме такий вигляд (після крапки з комою в кожному рядку наведено коментарі до заданої дії):

```
МОУ АХ. ШВ    ;прочитати число за адресою «ОКОВ 1 записати його до регістра АХ
МОУ ВХ, И0К0С  :прочитати число за адресою ШТОС [1] записати його до регістра ВХ
А00 АХ, ВХ     ;додати числа з регістрів АХ і ЗХ, суму записати в АХ
МІН М0К00      ;помножити вміст регістру АХ на число, записане за адресою ЬОР.ОО
МОУ ШК0С, АХ  ;записати число з регістра АХ за адресою ШР.0С
```

З прикладу видно, що навіть проста програма, написана мовою асемблера, складається з довгої послідовності команд, за структурою близьких до машинних. Написати таку програму нелегко, до того ж потрібно знати дуже багато подробиць щодо устрою комп'ютера (наприклад, для чого призначено ті чи інші регістри, які адреси пам'яті можна використовувати, а які — ні). Тому програмування мовою асемблера — справа окремих програмістів.

Для прискорення процесу програмування були розроблені *мови програмування високого рівня,* які дозволяли писати програми, за формою близькі до людської мови, та використовували загальноприйняту математичну нотацію. Перша з них з'явилася наприкінці 1950-х років і називалася Ротігап Назва була скороченням від слів РОКлшІа ТКАШайоп, що у перекладі означає трансляція формул. Опис мовою високого рівня програми, наведеної у прикладі 1.12, може виглядати так:

```
г=Ь+с
а=г*сі
```

Нагадаємо, що обчислюється вираз за формулою *а = (Ь* + с) х й. У тексті програми іменами а, Ь, с, сі, г позначені комірки пам'яті, в які записуються числа.

Одночасно з мовами високого рівня розроблялися *транслятори (компілятори)* — програмні засоби, призначені для перекладу високорівнєвих програм у машинні. Досвід створення мов високого рівня та їх трансляторів з роками накопичувався, Зокрема, було розроблено математичні основи та технологію реалізації цих програмних засобів. На сьогоднішній день кількість мов програмування й трансляторів вимірюється уже тисячами і продовжує зростати.

1.7.2. Технологія створення програми

Розглянемо процес створення програми у найбільш загальних рисах.

Розробка програми починається з постановки задачі, яку пропонує замовник. Іноді аналіз і уточнення задачі дають можливість формалізувати її постановку, в результаті з'являється математично точний і однозначний опис задачі. Після уточнення постановки задачі починається *проектування програми.* Як правило, в задачі можна виділити декілька *підзадач* і описати процес їх розв'язування окремо. Відповідно й алгоритм складається зі зв'язаних та узгоджених між собою частин, які описують процес розв'язання підзадач. У початковому алгоритмі дії подано в абстрактному вигляді, далекому від того, що може виконувати комп'ютер. Алгоритм уточнюють декілька разів і надають йому вигляд, за яким легко написати програму або її частину.

Написання програми або окремих її частин прийнято називати *кодуванням,* або *розробкою.* Найчастіше програму записують однією з мов високого рівня, але іноді деякі ЇЇ частини записують різними мовами. Далі програма перекладається (транслюється) на машинну мову (зазвичай, частинами). Під час кодування програмісти можуть припускатися помилок. Процес виявлення й виправлення помилок називається *налагодженням* програми. Він дозволяє виявити помилки перелічених нижче типів.

1. Помилки, пов'язані з порушенням правил граматики в тексті програми, написаної мовою високого рівня. їх можна виявити у процесі трансляції, тому вони називаються *помилками часу трансляції* (сошріїег етгог).
2. Помилки, що виявляються під час виконання робочої програми. Вони можуть виникати, наприклад, в результаті переповнення розрядної сітки чи при спробі видобути квадратний корінь із від'ємного числа. Такі помилки називаються *помилками часу виконання* (тип їіте еггог).
3. помилки, що не виявляються ні під час трансляції, ні під час виконання програми. Це змістові помилки, пов'язані з некоректністю логічних умов, неправильним використанням розрахункових формул і т. ін. їх називають *семантичними.*
4. Помилки у вихідних даних.

Налагодження - це процес багаторазового виконання програми з різними варіантами даних, які вона має обробляти. Дані спеціально добираються таким чином, щоб можна було виявити якнайбільше помилок, якщо такі існують. Ця цілеспрямована перевірка працездатності програми називається *тестуванням.* Тестування не гарантує відсутності помилок у програмі, а лише дозволяє виявити деякі з них. Чим ретельніше проводиться тестування, тим більше помилок виявляється та виправляється.

Після налагодження програма проходить *дослідно-виробничу експлуатацію,* для якої необхідно розробити супровідні документи під назвами «Керівництво розробника програми» і «Керівництво користувача», які описують устрій та використання програми. Перший документ дає можливість виправляти помилки під час експлуатації програми та розвивати її надалі, а у другому пояснюється, як використовувати програму.

Проте може з'ясуватися, що під час проектування програми було обрано не найкращий алгоритм, через що програма виконується надто повільно або витрачає забагато ресурсів пам'яті, тобто є неефективною. До програми нерідко вносяться зміни, які вимагають повторного кодування і налагодження. Якщо у замовника з'являються нові ідеї, необхідно заново ставити задачу та проектувати програму.

З метою розробки ефективних з точки зору використаних ресурсів програм і прискорення процесу їх проектування були створені технології, тобто системи методів, які дозволяють «не робити зайвих кроків» на кожному з етапів, від аналізу задачі до налагодження програми. У 70-х роках минулого століття домінувала *технологія структурного програмування* — система правил створення програм, що характеризуються ясністю, простотою тестування та налагодження, легкістю модифікації. Починаючи з 90-х років ключовою технологією є *технологія об'єктно-орієнтованого програмування* — програмування на основі абстрактних типів даних. Застосування різних технологій потребує постійного удосконалення інструментів, які дозволяють створювати програми швидко, якісно й економічно. Такі інструменти називаються *системами програмування* і реалізують дуже важливий *принцип повторного використання коду* завдяки створенню модулів і компонентів. У процедурному програмуванні принцип повторного використання коду реалізується за допомогою процедур і функцій, які розглядаються в розділі 4.

1.7.3. Перетворення програми і система програмування

Розглянемо, як із програми, написаної мовою високого рівня, утворюється інша — машинна. Програму (вихідний текст) за допомогою спеціальної програми (вона називається *текстовим редактором)* найчастіше записують на диск у вигляді вихідного файла (рис. 1.7). Програма може складатися з кількох вихідних файлів — у великих програмах їх може нараховуватися десятки.

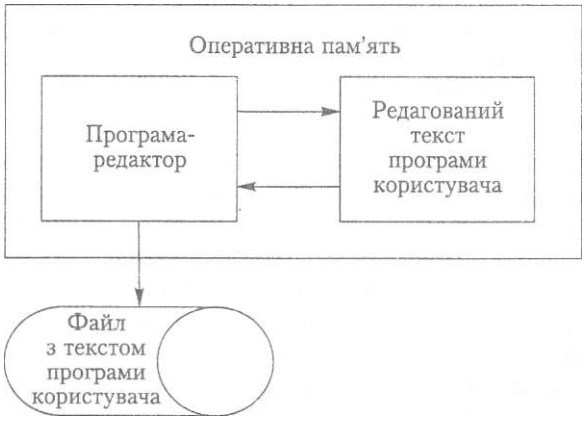


Рис. 1.7. Схема створення тексту програми

Під час роботи транслятора прочитується вихідний файл і створюється його машинний еквівалент — *об'єктний код.* Процес виконання програми-транслятора називається *трансляцією*, або *компіляцією* вихідного тексту.

Як правило, об'єктний код програми містить далеко не всі необхідні команди — програма може складатися з частин або включати підпрограми з бібліотек. Об'єктний код обробляється ще однією програмою - *редактором зв'язків,* або *компонувальником,* яка «збирає» (компонує) повний код програми і записує (завантажує) його або в оперативну пам'ять, або на диск у вигляді готового до виконання файла (рис. 1.8), який можна завантажити пізніше.

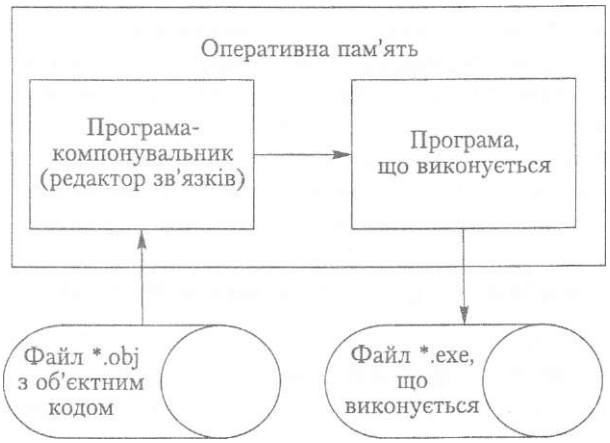


Рис. 1.8. Схема створення виконуваної машинної програми

Інтерпретатор на відміну від транслятора не створює машинну програму. Вхідні дані для інтерпретатора - це високорівнева програма й дані, що мають зчитуватися під час її виконання (рис, 1.9). *Інтерпретація* програми полягає в тому, що дії, задані програмою, відразу виконуються. Зазвичай інтерпретація вихідної програми відбувається повільніше, ніж виконання відповідної машинної програми.

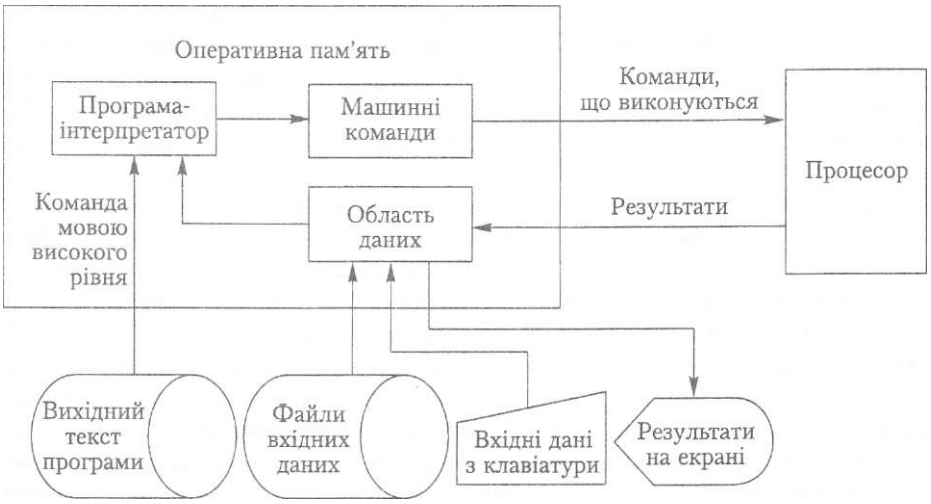


Рис. 1.9. Інтерпретація високорівневої програми

Ще один спосіб обробки вихідної програми поєднує в собі трансляцію й інтерпретацію. Програма перекладається (транслюється) не в машинні команди, а в деяке проміжне зображення, що потім інтерпретується. Такий підхід реалізовано, зокрема, в мові ,)ауа, яка швидко знайшла собі багатьох прихильників серед програмістів.

Інтерпретація програми здійснюється за допомогою такого інструменту, як *налагоджувач.* Він забезпечує інтерпретацію вихідної програми невеликими порціями (кроками) і дає можливість побачити результати виконання кожного кроку. Це полегшує пошук помилки (її локалізуй[1]'ю) у вихідній програмі.

Описані засоби (текстовий редактор, транслятор та (або) інтерпретатор, компонувальник, завантажувач і налагоджувач) разом утворюють *систему програмування,* або *інтегроване середовище розробки* (Іпіе§гаіесі Оеуеіортепі Епуноптепі, ГОЕ). Крім них до складу ШЕ входить бібліотека стандартних підпрограм, які можна використовувати під час створення програми.

1.7.4. Походження та розвиток мови Pascal

Повернімося до історії. Майже одночасно з мовою Рогігап було розроблено і реалізовано мовою АІ§о1 60 (АІ§огуіЬшіс 1ап§иа§е - алгоритмічна мова). Її конструкції були набагато більше схожі на англійські фрази, ніж конструкції Рогігап, тому логіка дій виражалася набагато природніше. На відміну від Рогігап, мова АІ§о1 мала засоби реалізації рекурсії, за допомогою якої легко записуються численні алгоритми. Завдяки цим і деяким іншим властивостям Аі^оі стала застосовуватися для запису алгоритмів, призначених для вивчення людиною.

У середині 1960-х років на основі мови Рогігап було створено мову Вазіс (Ве§іппег'з АІІ-ригрозе ЗушЬоІіс Іпзїгисііоп Сосіе — універсальний набір символьних команд для початківців). Вазіс була простішою за Рогігап і дозволяла легко та швидко створювати нескладні програми, але не підтримувала структурне програмування і тому не використовувалася під час створення великомасштабних проектів.

Мову Разсаі створив швейцарський учений Ніклаус Вірт спеціально для вивчення структурного програмування. Перший її опис було опубліковано в 1971 році. Вона походила від А1§о1 60 і двох мов на її основі, розроблених протягом 1960-х років (АІ§о1 \У та АІ§о1 68). Із середини 1970-х років Разсаі стала основною серед мов, які вивчаються першими.

Найбільш поширені версії систем програмування на основі цієї мови для машин типу ІВМ РС і сумісних із ними почали розроблятися фірмою Вогіапсі Іпіегпаїіопаї у 1983 році і дістали назву ТигЬо Разсаі. Версії з номерами до 4.0 реалізували стандарт мови Разсаі з незначними розширеннями. У четвертій версії (1987 рік) було істотно змінено технологію організації модульної структури програм. До складу цієї версії було включено вбудоване інтегроване середовище розробки (Іпїе^гаіесі Оеуеіортепі Епуігоптепі, ШЕ). Починаючи з версії 5.5 (1989 рік) мова концептуально розширена, тобто доповнена засобами об'єктно-орієнтованого програмування. Версія 6.0 постачалася разом із об'єктною бібліотекою ТигЬо Уізіоп і мовою Оіу'есї Разсаі, вбудованим асемблером (ВАЗМ), удосконаленими засобами налагодження в рамках нового ГОЕ й іншими доповненнями. Найбільш

відомою є система програмування ТигЬо Разсаі 7.0, створена фірмою Вогіаші Іпгегпаііопаї ще в 1993 році, а також її розширена версія — Вогіапсі Разсаі 7.0.

З 90-х років на основі мови ОЬ^есї Разсаі почала розвиватися ОеІрЬі — потужна система програмування, яка використовується для професійної розробки великомасштабних проектів. Вона є однією з найпопулярніших систем програмування, що забезпечують так звану швидку розробку програм (Карісі Арріісаііоп Оезі§п, К.АО). Ці системи забезпечують візуалізацію процесу створення програм і дозволяють істотно підвищити ефективність роботи програмістів.

Звичайно, більшість реальних програм створюються за допомогою систем програмування, в основу яких покладено не Разсаі, а інші мови. Найбільш поширеною серед професіоналів є мова С++, продовжує набирати прихильників мова ^уа, свої сфери застосування мають такі мови, як Рогігап, Вазіс та СОБОР (Сотшоп Визіпезз Огіепіесі Ьап§на§е - універсальна мова, орієнтована на розв'язання бізнес-задач) у їхніх сучасних версіях. Проте протягом професійної кар'єри програмістам неминуче доводиться освоювати декілька мов і систем програмування, а починати, на думку багатьох спеціалістів, краще все-таки з мови Разсаі.

1.8. Поняття алгоритму й основні алгоритмічні структури

Одним з базових понять інформатики є поняття алгоритму. Алгоритм вказує, які операції, пов'язані з обробкою даних, і в якій послідовності треба виконати, щоб отримати розв'язок задачі. Алгоритм розрахований на певного виконавця, з погляду котрого вказівки мають бути елементарними, тобто такими, що можуть бути виконані безпосередньо, без подальшого тлумачення. Слово «алгоритм» походить від назви латинського перекладу трактату арабського математика IX століття Аль-Хорезмі («Трактат Аль-Хорезмі про арифметичне мистецтво індусів»),

1.8.1. Властивості та способи опису алгоритму

Алгоритм має задовольняти певним вимогам, серед яких потрібно виділити найважливіші.

- *Визначеність* — кожен крок алгоритму має інтерпретуватися виконавцем однозначно.
- *Результативність* - за скінченну кількість кроків алгоритм має приводити до розв'язання задачі або зупинятися через неможливість її розв'язати.
- *Дискретність* — кроки обчислювального процесу мають бути відокремлені один від одного.
- *Ефективність* — під час розв'язання задачі може використовуватися лише обмежений обсяг комп'ютерних ресурсів.
- *Масовість* — алгоритм розробляється у загальному вигляді, тобто його можна застосувати не лише до окремої задачі, але і до деякого класу задач, що розрізняються лише вхідними даними. При цьому вхідні дані мають належати деякій області, яка називається *областю застосовності алгоритму.*

Є декілька способів опису алгоритму: словесний опис послідовності дій, алгоритмічна мова, аналітичний опис у вигляді набору формул, графічний — у вигляді блок-схеми тощо. Формалізована система правил текстуального опису алгоритмів є одним із різновидів мов програмування. А мови, призначені для запису алгоритмічних структур, називаються *мовами структурного програмування.*

Приклад 1.13_

Нехай квадратне рівняння *ах? + Ьх + с* - 0 задане дійсними коефіцієнтами *а, Ь, с* за умови, що *а Ф* 0. Розглянемо задачу обчислення дійсних коренів цього рівняння. Послідовність операцій, які необхідно виконати, є такою.

1. Прочитати коефіцієнти *а, Ь, с.*
2. Обчислити дискримінант *сі = Ь*^{2} - 4*ас.*
3. Якщо *с1>* 0, обчислити x_x = *(-Ь -у Г3) / 2 а ,* x_2 = *(-Ь + -/3)/2а* і написати ці числа; якщо *сі* = 0, обчислити *х = —Ь/2а* і написати це число; інакше написати «дійсних коренів немає».

Отже, у нас є задача - обчислити дійсні корені квадратного рівняння, заданого коефіцієнтами, і є опис процесу розв'язування задачі, або алгоритм. За цим описом ми виконуємо певну послідовність дій, тобто розв'язуємо задачу. Наведений приклад ілюструє словесний спосіб опису алгоритму.

Однією з наочних форм зображення алгоритму є *блок-схема.* Вона містить блоки, позначені геометричними фігурами. Усередині блоків записують елементарні дії. Блоки з'єднуються стрілками — так задається послідовність дій. Стрілки не є обов'язковими, якщо їхній напрямок відповідає просуванню «униз» і «праворуч». Кожній геометричній фігурі відповідає певний клас алгоритмічних інструкцій.

Зокрема, прямокутниками позначаються *операторні блоки.* Операторний блок може мати декілька входів і тільки один вихід. Це забезпечує однозначність у визначенні послідовності виконуваних дій. Дії, що позначаються такими блоками змінюють значення, форму подання чи розташування даних.

Ромбами позначається перевірка умови, залежно від результату якої визначається напрямок подальших обчислень. Тому блоки, що позначаються ромбами, називаються *умовними.* Оскільки результатом перевірки умови, записаної в умовному блоці, може бути значення «так» або «ні», тобто «істина» чи «хибність», блок має два виходи.

Алгоритм виконується у зовнішньому середовищі, де перебувають користувачі алгоритму. Від користувачів надходить інформація, яка в той чи інший спосіб оброблятиметься алгоритмом. Користувачі також повинні мати можливість отримати результати роботи алгоритму. Тому виникає потреба у блоках введення і виведення даних. Такі блоки позначаються паралелограмами.

Заокругленими прямокутниками позначається початок та кінець алгоритму. Початковий блок не має входів, а кінцевий блок - виходу. Обмежень на геометричні розміри блоків не існує.

Є три елементарні алгоритмічні структури: послідовності, розгалуження та повторення. Всі інші алгоритмічні структури утворюються з елементарних шляхом

заміни операторних блоків елементарними структурами. *Алгоритмічна структура послідовності* — це послідовність двох операторних блоків. Така структура дає вказівку виконувати одну інструкцію після іншої. Алгоритмічні структури розгалуження та повторення детально розглядатимуться в двох наступних розділах.

1.8.2. Алгоритмічна структура розгалуження

Алгоритмічна структура, що дозволяє виконавцеві алгоритму вибрати сценарій подальших дій залежно від істинності певного умовного твердження, називається *розгалуженням.* На блок-схемі (рис. 1.10) структури розгалуження позначаються ромбами. Дві стрілки, які відгалужуються від ромба, позначені словами «Так» і «Ні». Якщо записане всередині ромба умовне твердження є істинним, виконуються дії, на які вказує стрілка, позначена словом «Так». Якщо це твердження є хибним, виконуються дії, на які вказує стрілка, позначена словом «Ні».

Є декілька різновидів структури розгалуження. Структура, використана в алгоритмі обчислення коренів квадратного рівняння, є *альтернативним розгалуженням.* Альтернативне розгалуження припускає вибір виконавцем одного з двох можливих сценаріїв подальших дій залежно від істинності деякого умовного твердження. Крім альтернативного розгалуження є ще розгалуження у формі *множинного вибору альтернатив.* За множинного вибору може існувати більше двох сценаріїв дій виконавця. Вибір сценарію обумовлюється значенням деякого виразу.

1.8.3. Алгоритмічна структура повторення

Розглянемо задачу знаходження найбільшого спільного дільника двох натуральних чисел, застосувавши для її розв'язання алгоритм Евкліда. Позначимо найбільший спільний дільник чисел *а* і *Ь* через НСД(а, *Ь),* а остачу від ділення *а* на *Ь* - через *а* тосі *Ь.* Алгоритм Евкліда ґрунтується на тому факті, що НСД(а, *Ь*) = = НСД(6, *а* тосі *Ь),* якщо *Ь Ф* 0, і НСД(а, *Ь) = а,* якщо *Ь* = 0. Наприклад:

НСД(12, 5) = НСД(5, 12 то<і 5) = НСД(5, 2)=НСД(2, 5 тос! 2) = НСД(2, 1) =
= НСД(1, 2 тосі 1) = НСД(1, 0) = 1.

Алгоритм Евкліда вкрай незручно записувати за допомогою вказівок присвоєння та розгалуження, оскільки кількість таких вказівок залежить від значень *а* : Ь, тобто не є відомою наперед. Ефективний спосіб розв'язання цієї задачі полягає у застосуванні структури *повторення* (яка називається ще *циклічною структурою).* Алгоритмічна структура повторення дає виконавцеві алгоритму вказівку повторювати деякі дії, поки певне умовне твердження істинне.

Твердження, істинність якого перевіряється під час виконання циклічної структури, на блок-схемі записується всередині ромба (як і у випадку структури розгалуження). Особливістю зображення циклічної структури на блок-схемі є те, що одна зі стрілок повинна «повертатися назад», тобто має утворюватися замкнений «цикл» із блоків та стрілок. Такий цикл має містити умовний блок, в якому записана умова продовження повторення. Одна зі стрілок, що відгалужуються від цього блоку, повинна брати участь у циклі, а інша — вказувати на блок поза циклом.

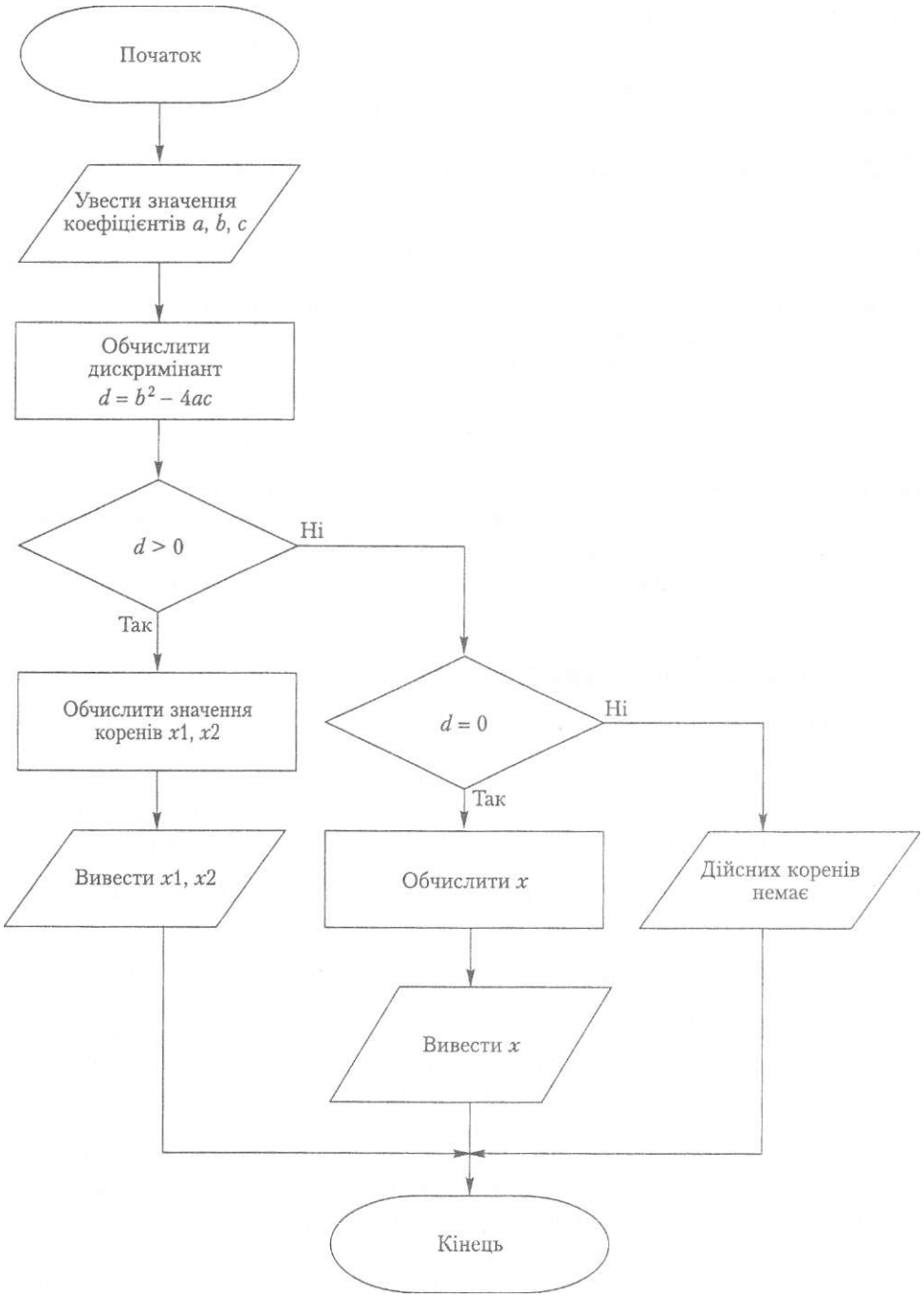


Рис. 1.10. Блок-схема алгоритму обчислення коренів квадратного рівняння

Приклад 1.14_

Використовуючи структуру повторення, опишемо алгоритм Евкліда для знаходження найбільшого спільного дільника двох натуральних чисел.

1. Прочитати значення *a* та *Ь*.
2. Поки *Ь Ф* 0, виконувати дії, описані у пунктах 3-5.
3. Обчислити величину *с* = *а* тосі *Ь*.
4. Значення *а* замінити значенням *Ь*.
5. Значення *Ь* замінити значенням с.
6. Написати значення *а*.

Структура повторення реалізується кроками 2-5. Виконання описаних на кроках 3-5 дій повторюватиметься доти, доки істинним є твердження *Ь Ф* 0. Істинність цього твердження перевіряється на кроці 2. Така перевірка здійснюється кожного разу перед тим, як виконуються кроки 3-5. Коли твердження *Ь Ф* 0 стане хибним, кроки 3-5 будуть пропущені і після кроку 2 буде виконано одразу крок 6.

Блок-схема алгоритму Евкліда зображена на рис. 1.11.

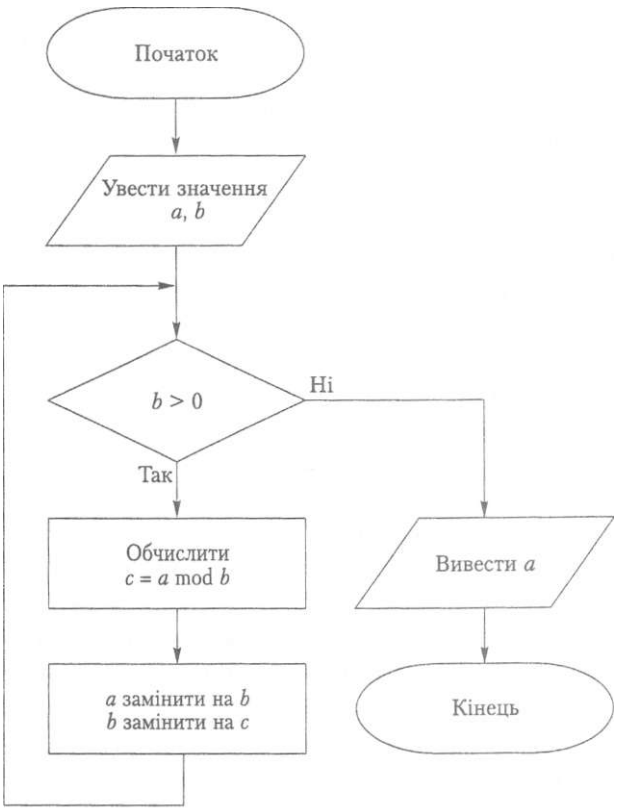


Рис. 1.11. Блок-схема алгоритму Евкліда

Висновки

- Принципи програмного керування фон Неймана визначають ідеологію архітектури електронних обчислювальних машин.
- У комп'ютері числа зображуються у двійковій системі числення цифрами 0 та 1, які кодують два різних стійких стани елемента пам'яті, що називається бітом.
- Вісім послідовних бітів утворюють байт. Байт є базовою одиницею виміру довжини інформаційного повідомлення й обсягу пам'яті.
- Оперативна пам'ять може розглядатись як послідовність байтів. Кожен байт має свій номер — адресу.
- З погляду процесора програма — це розташована в оперативній пам'яті послідовність команд. Команди виконуються арифметико-логічним пристроєм процесора.
- Мови програмування високого рівня дозволяють записувати програми в зрозумілих для людини термінах.
- Переклад програми, що записана мовою високого рівня, на машинну мову (в об'єктний код) називається трансляцією і здійснюється програмою-транслятором.
- Програма-інтерпретатор виконує визначені програмою дії, не транслюючи програми у машинний код.
- Редактор зв'язків, або компонувальник, створює повний код програми шляхом поєднання об'єктного коду її окремих модулів.
- До складу інтегрованого середовища програмування входять текстовий редактор, транслятор та (або) інтерпретатор, компонувальник і налагоджувач.

4- Розробка програмного забезпечення - це складний ітеративний процес, що, як правило, складається з таких етапів: постановка задачі, аналіз задачі та побудова її моделі, вибір або розробка алгоритму розв'язання задачі, кодування, налагодження та тестування, дослідно-виробнича експлуатація і супровід програмного забезпечення.

+ Алгоритм — це набір елементарних вказівок, розташованих у певній послідовності. Алгоритм розрахований на певного потенційного виконавця, з погляду якого вказівки повинні бути елементарними, тобто такими, що можуть бути виконані безпосередньо, без подальшого тлумачення.

- Розгалуження — алгоритмічна структура, яка дозволяє виконавцеві алгоритму вибрати сценарій подальших дій залежно від виконання певних умов.

+ Різновидами розгалуження є альтернативне розгалуження, що припускає вибір виконавцем одного з двох можливих сценаріїв подальших дій, та множинний вибір альтернатив, за якого таких сценаріїв може бути більше двох.

+ Алгоритмічна структура повторення дає виконавцеві алгоритму вказівку повторювати деякі дії доти, доки певне умовне твердження є істинним.

Контрольні запитання та завдання

1. Дайте означення поняття «архітектура комп'ютера».
2. Сформулюйте основні принципи фон Неймана.
3. Які функціональні блоки входять до складу комп'ютера?
4. У чому полягає призначення процесора та його регістрів?
5. Чим відрізняється кеш-пам'ять від інших типів оперативної пам'яті?
6. Які компоненти входять до складу зовнішньої пам'яті?
7. Що таке машинна команда?
8. Що таке позиційна система числення і як знайти значення числа за його записом у певній позиційній системі?
9. Чому інформація в комп'ютері записується у двійковій системі числення?
10. Для чого використовується шістнадцяткова система числення?
11. Як перевести десяткове число до будь-якої іншої системи числення?
12. Вкажіть основні форми зображення чисел у комп'ютері.
13. Що таке біт та байт?
14. Що таке програма?
15. Чим мова програмування високого рівня відрізняється від машинної мови?
16. Опишіть процес створення програми.
17. Що таке транслятор?
18. Чим відрізняється компіляція від інтерпретації?
19. Чим мова Разсаі відрізняється від мов Рогігап і Вазіс?
20. Перелічіть складові інтегрованого середовища розробки програм.

Вправи

1. Побудувати блок-схему для визначення типу трикутника (рівносторонній, **рів**нобедрений, різносторонній) за довжинами його сторін. Якщо трикутник не можна побудувати, слід вивести повідомлення.
2. Побудувати блок-схему алгоритму піднесення цілого числа до цілого степеня.
3. Побудувати блок-схему алгоритму переведення цілого числа з десяткової **сис**теми числення до будь-якої іншої.
4. Побудувати блок-схему алгоритму переведення числа з будь-якої системи числення у десяткову.
5. Побудувати блок-схему алгоритму обчислення факторіала натурального **числа.**
6. Побудувати блок-схему алгоритму розв'язання такої старовинної задачі. **Три** лицарі та їх зброєносці мають переправитися на інший берег річки. Човен **може** вмістити двох осіб. Як їм переправитися за умови, що без свого лицаря **жоден** зброєносець не може перебувати у товаристві інших лицарів.

7. Записати алгоритм обміну значеннями між двома однотипними змінними за умови, що:
   а) можна використовувати третю змінну;
   б) третю змінну використовувати не можна (змінні числові).
8. Записати алгоритм «циклічного» обміну значеннями між трьома змінними *a, Ь, с (a<—Ь, Ь<—c,* c<—*a).*
9. Нехай г — числова змінна. Використовуючи лише операції множення та вказівки присвоєння, записати алгоритм обчислення таких значень: *г , г* , 2^{15}, 3^{31}. Операцій множення має бути якомога менше. Наприклад, обчислення 2^6 можна задати так: 2^=2x2; 2^4:=2^2x$г^2$; 2^6:=2^4x$г^2$. Як бачимо, тут лише три операції множення замість п'яти, що виконуються при тривіальному способі обчислення: 2^6 = 2 x 2 x 2 x 2 x 2 x 2.
10. Перевести десяткові числа 100, 255, 256, 640,1024, 32767 до двійкової та шістнадцяткової систем числення.
11. Перевести двійкові числа 1000, 1111, 110 0100, 1111 1111, 1 0000 0000 до десяткової та шістнадцяткової систем числення.
12. Подати шістнадцяткові числа Р1, РР, 4АВ та РРРЕ у десятковій та двійковій системах числення.
13. Подати 36-кові числа 2У та 100 у десятковому записі (36-кові цифри А, В,..., У, 2 позначають десяткові числа 10, 11, ..., 34, 35 відповідно).
14. За основою Р та *Р*-ковим записом дробу вказати його десяткове зображення:
   а) Р = 2; 0,0001; 0,1111; 0,11111111;
   б) Р = 3; 0,001; 0,22; 0,11;
   в) Р = 16; 0,1; 0,РР; 0,8; 0,(7).
15. Записати Р-кове зображення десяткового дробу *ci,* де:
   а) ◁ = 0,5, Р = 2, 3, 5,8, 16, 20;
   б) *ci* = 0,1, Р = 2, 3, 5, 8, 16, 20.
16. Указати двобайтовий додатковий код чисел -1, -8, -9, -32767, -32768.
17. Припустимо, що при додаванні та відніманні чисел перенесення зі старшого розряду стає вмістом знакового розряду, а перенесення зі знакового розряду втрачається. Вказати значення виразу (величини *maxi* та *mïni* позначають максимальне та мінімальне цілі числа):
   а) *maxi* + 1;
   б) *mini* - *1.*
18. Обчислити мінімальне та максимальне за модулем скінченні дійсні числа, що зображуються за допомогою:
   а) 4 байтів з *ci* = 8, *г* - 23;
   б) 8 байтів з *ci* = 11, *г* = 52;
   в) 10 байтів з *ci* = 16, *г* = 63.

   Тут *г* — розрядність мантиси, а *ci* - розрядність порядку.

Розділ 2

Елементи мови PascaI

- Процес створення та виконання найпростішої програми у середовищі Borіапсі PascaI 7.0
- Алфавіт, лексичні одиниці та загальна структура програми
- Поняття типу даних, константи, виразу, змінної
- Прості типи даних
- Операції у мові PascaI
- Оператор присвоєння та виклику процедур введення-виведення

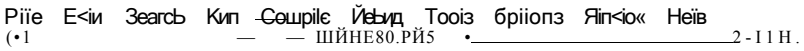
2.1. Робота у середовищі Borіапсі PascaI 7.0

Інтегроване середовище розробки Borіапсі PascaI 7.0 - далі ШЕ (Іпіе§гаіесі 1)е\'е-Іоршепі Епуігопніепс) Borіапсі PascaI **7.0** - складається з текстового редактора, компілятора, компоиувальника, налагоджувана і довідкової системи. Стандартна поставка ШЕ Borіапсі PascaI **7.0** являє собою дистрибутивний набір файлів, що містить файл іп5іаII.exe, який слід виконати для установки інтегрованого середовища. За замовчуванням установка здійснюється до каталогу **C:\BP.** До складу ГОЕ Borіапсі PascaI **7.0** увійшли три версії компілятора:

- компілятор, який працює під керуванням операційної системи МЗ-ООЗ в реальному режимі процесора (файли **иігЬо.ехе, фс.ехе)** і генерує МЗ-БОЗ-програми;
- компілятор, який запускається під керуванням операційної системи МЗ-БОЗ в захищеному режимі процесора (файли **Ьр.ехе, Ьрс.ехе) і** генерує програми, що працюють як в МЗ-ВОЗ, так і у **\Уіпсіо\У5;**
- компілятор, що працює під керуванням операційної системи \¥іпсіо\¥5 (файл Ьрм.ехе) і генерує \¥ін<іо\¥з-програми.

За допомогою файлів **ІигЬо.ехе, Ьр.ехе** і **Ьрм/.ехе** активізується інтегроване середовище розробника, яке відображається на екрані у вигляді вікна (рис. 2.1, 2.2). Файли ірс**.ехе** і Ьрс**.ехе** дозволяють запустити компілятори, що працюють у режимі командного рядка. У цьому разі ГОЕ Borіапсі PascaI 7.0 не завантажується.

Створюючи програму в ГОЕ Borіапсі PascaI **7.0,** програміст має можливість користуватися довідковою інформацією, бібліотекою стандартних підпрограм, драйверами для виконання графічних програм тощо. З цією метою до пакету Borіапсі PascaI **7.0** включені файли довідкової системи **(іхігЬо.ірїі, Ьр\п/.Ипр),** бібліотек стандартних модулів **(иігЬо.ірІ, ірр**.ірі, **ір\п/.ірІ),** конфігурації ГОЕ Borіапсі PascaI **(иігЬо.ір,** Ьр.ір, Ьрм.сід) і багато інших.



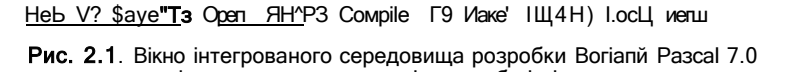
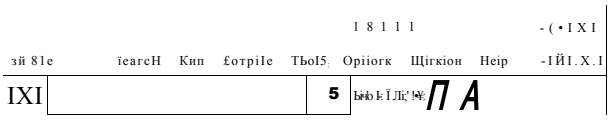


Рис. 2.1. Вікно інтегрованого середовища розробки Borlapй Pascal 7.0 після запуску програми Ьр.exe або IигЬo.exe



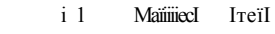


РИС. 2.2. Вікно інтегрованого середовища розробки Borlapcі Pascal 7.0 після запуску програми Ьрс.exe

Розглянемо технологію використання ШЕ Borlapcі PascaI 7.0. Основне вікно ШЕ Borlapcі PascaI 7.0 (рис. 2.1) складається з таких функціональних частин, як рядок меню, робоча зона та рядок стану. Меню активізується натисканням функціональної клавіші Р10 або шляхом вибору мишею певного розділу меню, що може бути активізований також комбінацією клавіш АЇ+перша літера його назви. Розглянемо призначення розділів головного меню.

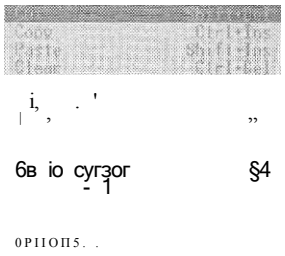
4- РІІе — містить команди управління файлами (створення нових і завантаження наявних файлів, збереження файлів на дисках, виведення вмісту файла на принтер, вихід з ШЕ Borlapcі PascaI 7.0).

- Ecііі: — дає можливість виконувати команди редагування тексту (копіювання, вставляння, видалення фрагментів тексту, відновлення попереднього варіанта тексту, що редагується).
- ЗеагсЬ — призначений для виконання пошуку фрагмента тексту та його заміни новим фрагментом.
- Кип - дає можливість запускати програму в цілому або виконувати її по кроках (останній режим застосовується при налагодженні).

4 Сотріїе - дає можливість компілювати програму.

- ОеЬид - містить команди, які полегшують процес пошуку помилок (установка точок переривання виконання програми, виведення вікна перегляду значень змінних, вікна вихідних результатів тощо).
- тооїз - містить перелік допоміжних інструментів, що не включені до ШЕ Borlandcі Pascal 7.0 (ТигЬо АззетЬІег, ТигЬо БеЬи§§ег тощо).
- ОрИопз — дає можливість встановлювати директиви компілятора та компонувальника, а також управляти параметрами середовища Borlandcі Pascal 7.0.
- Міпсіои — містить команди керування вікнами.
- Неір — дає можливість отримувати довідкову інформацію.

В ШЕ Borlandcі Pascal 7.0 є також контекстне меню (рис. 2.3), яке містить команди, що найчастіше використовуються. Відкрити контекстне меню можна за допомогою комбінації клавіш АІІ+Р10 або правою кнопкою миші. Натиснувши клавішу ЕБС, можна вийти з контекстного меню.



В Нвір ГСєтоуєіНе хеіесіес! ієхі ага риі Н іо іЬо ClipЬоагй

РИС. 2.3. Контекстне меню ЮЕ Borlandcі Pascal 7.0

Робоча зона (Безкіор) — це простір, у якому можуть бути розташовані декілька вікон. Кожне вікно має заголовок і номер. За замовчуванням вікну з номером 1 залається заголовок MCMAME00.PA5. Вікно може мати довільний розмір і розташо- : уватися у будь-якому місці екрана. Робота з вікнами відбувається так само, як у будь-якому іншому багатовіконному середовищі. На правій і нижній межах

вікна містяться лінійки прокрутки, що дають можливість пересувати текст у вікні за допомогою миші. Використання клавіш управління курсором дозволяє зсувати текст на один рядок. Клавіші Раде Пр, Раде Ооип застосовуються для посторінкового гортання тексту. Вікно можна закрити комбінацією клавіш Аіі+Р3, кнопкою закриття вікна, що розташована у його лівому верхньому куті, а також командою **Шпсіоог** • СІ05Є.

Рядок стану міститься в нижній частині вікна ГОЕ Вогіапсі РаzcaI 7.0. В ньому наведені відомості про операції, що найчастіше виконуються, та надається перелік клавіш для їх швидкого виклику (такі клавіші називаються Ьоі кеуз - гарячі клавіші).

До першого збереження тексту з вікна ^^МЕОО.РАБ користувач має визначити каталог, у якому міститимуться файли його програм. Цей каталог зручно було б зробити поточним каталогом інтегрованого середовища. Поточний каталог розкривається командою Рііе Орєп, а команда Рііе • Бауе зберігає до нього файли текстів програм. Крім того, створені під час трансляції виконувані файли також записуватимуться до поточного каталогу, За замовчуванням поточним вважається каталог **\BP**\Віп, але збереження програм до цього каталогу є небезпечним, оскільки в ньому містяться системні файли Вогіапсі РаzcaI 7.0. Поточний каталог встановлюється командою Рііе • Сіїапде сііг. Програмісту слід створити свій каталог і зробити його поточним.

Створення та виконання найпростішої програми

Надрукуємо у вікні MONAMEOO.PA5 текст програми з прикладу 2.1. (інформація щодо команд текстового редактора міститься в довідковій системі ГОЕ Вогіапсі РаzcaI **7.0** у розділі ІІБіпд іНе есііюг).

Приклад 2.1

```
ргодгаш ex2_1;    {заголовок програми          }
Ьеділ             {початок опису дій програми}
  мгНеInCHeIIo, могісі! '); {вивести символи}
єпсі.             {кінець програми             }
```

Програму слід зберегти на диску перед її виконанням. Для цього використовується клавіша **Р2** або команда **Рііе** • 53УЄ. Під час першого збереження програми буде виведене вікно **5аує** Я!є аз. У рядку 5ауе іїіе аз задамо ім'я файла та натиснемо клавішу **Еніег.** Розширення **ра5** в імені файла можна не задавати, оскільки таке розширення для файлів текстів програм встановлене за замовчуванням (змінити розширення імен файлів, що встановлені за замовчуванням, можна командою Орйоп5 • **Епуігопгепі** • Есііюг). Подальші виклики команди Рііе • 5аує зберігатимуть програму автоматично, без виведення вікна 5аУЄ Яіе аз У разі потреби змінити ім'я або розташування файла програми це вікно можна вивести командою **Рііе** • **5ауе аз.** Періодичне натискання клавіші Р2 під час роботи з редактором забезпечить збереження всіх виправлень у тексті програми.

Після введення та збереження у файлі програму можна компілювати, скориставшись комбінацією клавіш АІІ+Р9 або командою меню Сотріїе • Сотрііе. Коли

компілятор виявить синтаксичну помилку, то виведе відповідне повідомлення і встановить курсор на рядок, що містить помилку. Зазначимо, що компілятор лише виявляє помилки, а виправляє їх розробник програми. Компілятор також не може знайти алгоритмічних помилок, відповідальність за логічну коректність програми покладено на програміста. Якщо компіляція завершилася успішно, то буде виведено вікно із повідомленням **Сотріїе зисссззГиі. Ргезз апу кеу** (Компіляція успішна. Натисніть будь-яку клавішу). У результаті успішної компіляції створюється ехе-файл, який має таке саме ім'я, як і раз-файл програми. Компіляцію разом із компонуванням можна виконати за допомогою команд меню **Сотріїе • Маке** і **Сотріїе • Вийсі.** Команда **Маке** призведе до перекомпіляції модифікованих модулів (використання модулів буде розглянуто у розділі 6). За допомогою команди **Вийсі** перекомпілюються всі модулі, які використовуються програмою.

Програму, в якій виправлено всі синтаксичні помилки, можна запустити на виконання комбінацією клавіш Ох!+Р9. Її також можна виконати за допомогою команди **Кип • Кип.** Результати роботи програми ви побачите, натиснувши клавіші АІІ+Р5 або виконавши команду ОеЬид • ІІзег зсгееп (рис. 2.4). Перегляд результатів виконання програми завершиться після натискання будь-якої клавіші. На екран буде виведено вікно із текстом програми, результати роботи якої користувач переглядав.

Рис. 2.4. Результати роботи програми ex2_1

Якщо помилка виникає під час виконання програми, то її робота переривається і виводиться повідомлення про помилку часу виконання (гап-ііте еггог). Програма може «зациклитися» — в такому разі постає потреба у примусовому перериванні її виконання. Для примусового зупинення програми використовується комбінація клавіш **Сігі+Вгеак.**

Щоб продовжити редагування програми, яка була раніше записана до файла, слід відкрити цей файл командою **Ріїе • Ореп** або натисканням клавіші **Р3.** У діалоговому вікні **Ореп** треба вибрати ім'я файла зі списку **Ріїез** і натиснути клавішу Епїег. У результаті цих дій у робочій області ШЕ Boгіапсі РаscaI 7.0 створиться нове вікно, що міститиме текст обраного файла. Ім'я цього файла відобразиться в заголовку вікна. Щойно відкрите вікно є активним, воно виводиться перед іншими вікнами та має рамку з подвійних ліній.

Комбінація клавіш СігІ+РБ. що викликає команду \А/іпсіо\м • **5І2Є/МОУЄ**, дозволяє змінити розміри вікон та їх розташування. В результаті виконання команди \Д/іпбом • **5І2Є/МОУЄ** рамка активного вікна набуде зеленого кольору. Це означатиме, що вікно перебуває у режимі модифікації. У цьому режимі за допомогою клавіш управління курсором здійснюють переміщення вікна. Розміри вікна можна змінити, скориставшись комбінацією клавіші 5Ьй із клавішами управління курсором. Для виходу з режиму модифікації зі збереженням внесених змін ми натискаємо

клавішу Enter, а для їх скасування - клавішу **Esc.** Вікно молена розгорнути на **всю робочу** область, натиснувши клавішу **F5.**

Завершення роботи з ШЕ Borland Pascal 7.0 здійснюється комбінацією клавіш **Alt+X** або командою File • Exit. При цьому користувачеві буде запропоновано зберегти всі файли, до яких було внесено зміни після останнього збереження.

Налагодження програм

Для виявлення алгоритмічних помилок до ШЕ Borland Pascal 7.0 вбудовано налагоджувач, функції якого реалізуються командами меню **Debug.** Такі помилки найлегше виявити при покроковому виконанні програми. Програма у покроковому режимі запускається командою Run • Trace Into або клавішею **F7.** За допомогою цієї ж клавіші виконується кожний наступний крок програми, якому відповідає один рядок операторів.

На кожному кроці значення використаних у програмі змінних можуть модифікуватися. Поточні значення змінних відображаються у вікні **Watches,** яке активізується командою Debug • Watch. Для перегляду значень певної змінної її ім'я слід вказати у вікні Add **Watch,** що активізується комбінацією клавіш **Ctrl+F7** або командою **Debug** • Add «ajcb. Якщо ім'я змінної було введене у вікні **Add Watch,** її поточні значення виводитимуться у вікні Watches (рис. 2.5). Коли вікно **Watches** активне, додати до нього змінну можна як за допомогою комбінації клавіш **Ctrl+F7,** так і натиснувши клавішу Insert, а для видалення змінної слід обрати її ім'я і натиснути клавішу Delete.

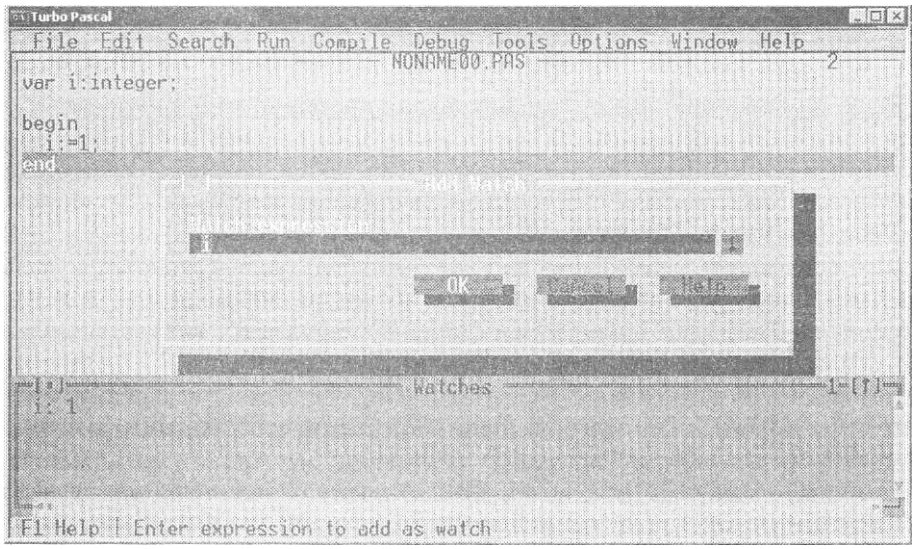


Рис. 2,5. Вікна Add Watch та Watches, призначені для перегляду значень змінних під час налагодження програми

Для завершення покрокового виконання програми використовується команда **Run • Program reset** або комбінація клавіш Ctrl+F2.

Якщо програмісту необхідно отримати значення змінних під час виконання певних операторів програми або певних умов, то в тексті програми можна встановити точки переривання програми (Ьгеакроіиїз). З цією метою використовують команду **РеЬид • Вгеакроіпіз.** Для кожної точки переривання у вікні **Вгеакроіпіз** можна задати номер рядка в тексті програми, умову, виконання якої зупиняє програму, та кількість проходжень через точку переривання до виконання умови. Задати точку та умову переривання можна також командою ОеЬид • **Асісі Ьгеакроіпі,** встановивши перед тим курсор на відповідний рядок у тексті програми. Рядок програми, який містить точку переривання, помічається у вікні текстового редактора червоним кольором. Програма, запущена на виконання командою **Кип • Кип,** досягнувши виділеного рядка, зупиниться в разі істинності умови переривання, **якщо** така умова задана.

2.2. Словник мови та загальна структура програми

У будь-якій мові програмування програма — це набір зрозумілих компілятору команд. Для створення програм треба знати синтаксис мови, тобто правила запису команд і використання лексичних одиниць мови. Знайомство з мовою розпочнемо з алфавіту.

2.2.1. Алфавіт і словник мови

Алфавіт мови програмування - це скінчений набір символів. За допомогою цих символів можуть бути записані ідентифікатори, вирази та оператори мови.

Алфавіт мови РазсаІ є підмножиною символів із кодової таблиці А5СІІ (від Атегісап Зіапсіагсі Сосіе іог ІніогтаГіоп ІиїегсЬап§е - Американський стандартний код обміну інформацією). Кожному такому символу відповідає числовий код від 0 до 255. Частина символів кодової таблиці А5СП з кодами від 0 до 127 ідентична для всіх ІВМ-сумісних комп'ютерів. Символи алфавіту мови РазсаІ можна поділити на такі категорії:

- символи, що використовуються для складання ідентифікаторів (малі латинські літери з кодами АЗСІІ від 97 до 122, великі латинські літери з кодами АЗСІІ від 65 до 90, десяткові цифри від 0 до 9 з кодами А5СІІ від 48 до 57, символ підкреслення (_) із кодом АЗСІІ 95);
- розділовий символ пробілу, код АЗСІІ 32;
- спеціальні символи, які використовуються у процесі побудови конструкцій мови (+ - * / = > < . , ; : ' ( ) [ ] { } Л ® $ #):
- керуючі символи, що мають АЗСІІ-коди від 0 до 31.

Із символів алфавіту складаються лексичні одиниці мови, або *лексеми* — мінімальні значущі одиниці в текстах програм. Множина всіх допустимих лексем

називається *словником* мови програмування. У мові PascaI розрізняють такі види лексем: спеціальні символи, зарезервовані (ключові) слова, ідентифікатори, неіменовані константи, коментарі та директиви компілятора.

Лексеми *спеціальних символів,* окрім спеціальних символів з алфавіту мови, містять ще складені спеціальні символи, що сприймаються компілятором як єдине ціле (<= >= := (* *) ..).

Зарезервовані (ключові) слова мають строго визначений зміст. їх призначення не може змінюватися. Зарезервовані слова використовуються для позначення алгоритмічних конструкцій, розділів програми тощо.

Ідентифікатор - це ім'я, значення якого може варіюватися від програми до програми або навіть у межах однієї програми. У мові PascaI розрізняють стандартні ідентифікатори та ідентифікатори користувача.

Стандартними ідентифікаторами є імена вбудованих у мову процедур і функцій (reaci, игіїе, зіп, соз тощо), типів даних (іпіедег, reai, сіїаг тощо) і директив ("Гогмагсі, уігіиаі, аЬзоІиІе тощо). Для стандартних ідентифікаторів припустимим є переозначення, при якому вони втрачають стандартний зміст у межах даної програми.

Ідентифікатор користувача — це ім'я, яке обирає програміст для позначення (ідентифікації) елементів програми (іменованих констант, змінних, типів, полів запису, процедур, функцій, модулів, програм). Існують правила запису ідентифікаторів:

- ідентифікатор починається буквою або символом підкреслення;
- ідентифікатор може складатися із букв, цифр, символу підкреслення;
- ідентифікатор може мати довільну довжину, але значущими є тільки перші 63 символи;
- в ідентифікаторі неприпустимо використовувати символи пробілу, крапки та інші символи пунктуації;
- малі та великі літери в ідентифікаторах не розрізняються;
- зарезервовані слова не можуть використовуватись як ідентифікатори.

Наведемо деякі рекомендації щодо використання імен у програмах:

- використовуйте мнемонічні ідентифікатори (такі, що легко запам'ятовуються);
- використовуйте довгі ідентифікатори, що складаються із декількох слів, кожне з яких починається з великої літери, наприклад: СотріЛегбгарНіcзІпіег'Гасе, Мпептапі cCocie;
- замість транслітерацій українських і російських слів бажано використовувати їх переклади англійською мовою.

Дотримання цих рекомендацій, нарівні із застосуванням коментарів, зробить текст програми більш зрозумілим.

Серед *неіменованих констант* можна вирізнити числа, символи, рядки та логічні константи. Детальніше різновиди констант розглядатимуться в підрозділі 2.4.1.

Коментар — це фрагмент тексту програми, який записується у фігурних дужках ({}) або вміщується між лексемами (* та *). Коментарі пояснюють призначення окремих фрагментів програми. Вони ігноруються компілятором і не впливають

на роботу програми, але полегшують її розуміння. Далі наведемо приклади коментарів:

```
{ Коментар записується у фігурних дужках }
(*Коментар записується у круглих дужках із "зірочками"*)
```

Директива компілятора — це коментар, у якому безпосередньо за фігурною дужкою заїшсано символ «$». Директиви компілятора визначають *режими компіляції* і можуть істотно впливати на зміст згенерованого компілятором машинного коду.

Деякі лексеми не повинні розташовуватися поруч. Наприклад, два поруч записаних ключових слова можуть інтерпретуватися компілятором як один ідентифікатор. Для відокремлення таких лексем застосовують *порожні символи,* або *пробіли,* — символи, що не мають зображення та з'являються у тексті програми як результат натискання клавіш Брасе (пробіл), Епїєг (кінець рядка) і **Tab** (символ табуляції). Пробіл між сусідніми лексемами не обов'язковий, якщо хоча б одна з них є роздільником (наприклад, символом крапки з комою), коментарем або знаком операції.

2.2.2. Структура програми

Записана мовою РазсаІ програма складається із двох частин: декларативної (оголошення ідентифікаторів, що використовуються у програмі) та операторної (запис виконуваних дій). Декларативна частина програми передує операторній і складається з розділів, кожен з яких починається певним ключовим словом. Операторна частина розпочинається ключовим словом Ьеділ і завершується ключовим словом єпсі. (із крапкою). Усередині операторної частини також можуть використовуватися слова Ьеділ та єпсі, але жодне з таких слів не повинне завершуватися крапкою. Наведемо послідовність частин і розділів Разсаі-програми.

```
ргодгат     {заголовок програми                    }
   }        {розділ директив компілятора          }
изез        {розділ підключення модулів           }
сопзї       {розділ оголошення іменованих констант}
їуре        {розділ оголошення типів              }
Vаг         {розділ оголошення змінних            }
ргосесіиге  {розділ оголошення процедур           }
ґипсїіоп    {розділ оголошення функцій
Ьеділ       {операторна частина                   }
            {оператори                            }
епгі.       {кінець програми                      }
```

Зауважимо, що всі розділи декларативної частини є необов'язковими, а такі розділи, як сопзї, іуре та уаг, можуть бути записані у довільному порядку довільну кількість разів.

На початку програми міститься її *заголовок*, що складається із зарезервованого слова ргодгат, імені програми та параметрів, за допомогою яких вона взаємодіє з операційною системою. Заголовок програми не є обов'язковим, його можна опустити. Він використовується з метою швидкої ідентифікації потрібної програми поміж інших.

У *розділі директив компілятора* наводиться список директив, які визначають режими роботи компілятора. Синтаксис оголошення директиви компілятора має такий вигляд:

```
{$<ім'я директиви^
```

В одному коментарі можна згрупувати декілька директив компілятора, розділивши їх комами, наприклад:

```
{ÏA+.И-}
```

Великі програми можуть складатися з декількох програмних одиниць - однієї головної програми та довільної кількості *модулів.* Усі вони зберігаються в окремих файлах, а також окремо компілюються. У результаті компіляції модуля створюється його об'єктний код, що підключається до програми під час компонування. Щоб компонувальнику було відомо, які модулі треба підключати, на початку головної програми зазначаються імена модулів. Перелік імен модулів міститься у *розділі підключення модулів* і починається із зарезервованого слова изез. У програмі може бути лише один розділ підключення модулів, він розташовується після заголовка програми та перед усіма іншими її розділами. Докладніше створення та використання модулів розглядатиметься в розділі 6. Синтаксис розділу підключення модулів такий:

```
изез <ім'я модуля1>, <ім'я модуля2>;
```

Розділи оголошення ідентифікаторів — це послідовності оголошень імен констант, типів, змінних, процедур і функцій. Кожне оголошення завершується символом крапки з комою (;).

УВАГА

Ідентифікатор має бути оголошеним до його першого використання в операторних частинах програми, процедур та функцій.

У *розділі оголошення іменованих констант* задають ідентифікатори констант та їх значення за таким синтаксисом:

```
сопзі: <їдентифікатор константи> = <значення константного виразу>:
```

Оголошені у розділі сопзї ідентифікатори набувають числових, символьних, рядкових або логічних сталих значень, або значень числових, рядкових, або логічних виразів. Ці значення не можуть змінюватися у програмі.

Розділ оголошення типів даних використовується тоді, коли користувач створює власні типи даних. Стандартні типи даних, що розглядатимуться в наступному підрозділі, не потребують оголошення. Синтаксис оголошення типів даних такий:

```
їуре <ідентифікатор типу> = <опис типу>:
```

На відміну від значень констант, значення змінних можуть змінюватися під час виконання програми. У *розділі оголошення змінних* вказуються імена всіх використаних в операторній частині програми змінних. Наведемо синтаксис оголошення змінної:

```
уаг <ідентифікатор змінної>:<тип>:
```

Різновиди оголошень ідентифікаторів розглядатимуться разом з уточненням відповідних понять (змінних, констант, типів, процедур і функцій). Принцип оголошення процедур і функцій розкривається у розділі 4.

Операторна частина програми (або її *тіло)* містить набір операторів, що визначають дії програми. Як вже зазначалося, тіло програми оточується *операторними дужками* — парою ключових слів Ьеdіп та єпсі. Після останньої операторної дужки єпсі обов'язково ставиться крапка. Розташований після цієї крапки текст ігнорується компілятором.

Розглянемо програму, в якій є розділи оголошення іменованих констант і змінних, а також операторна частина. Коментарі пояснюють призначення змінних і константи, а також операторів програми.

Приклад 2.2

Необхідно за введеним із клавіатури значенням температури за шкалою Цельсія визначити відповідне значення температури за шкалою Кельвіна. Обчислення виконуються за простою формулою:

$$TкеЬ = T_{Ce}i5 + 273.$$

Результати роботи програми зображено на рис. 2.6.

```
ргодгат ех2_2;
                          {декларативна частина        }
  с0п5ї
    Кєіус = 273;          {оголошення константи Кєіус }
  уаг
    їсеїз."ЬкеїV: 1 гтЬедег;  {оголошення змінних
                            Псе15, ПкеїV               }
Ьеdіп                     {операторна частина          }
                          {запросити користувача до введення числа}
  мгіїе1п( 'Сеїзіиз Іеглрегаїиге:');
  геасіїпСісеїз):         {ввести із клавіатури
                           ціле число та записати його в змінну Гсеїз}
  ФкеїV := ісеїз+Кєіус;   {обчислити суму значень
                           двох змінних, запам'ятати ії у змінній їкеїу.
                           вивести пояснення і значення змінної їкеїу   }
  мгіІе"Іп('Кєіуіп їетрегаїиге:', ІкеІУ);
єпсі.                     {кінець програми             }
```

```
Cpi5iii5 іепрегаїиге:  '   ' ^
3О                                  1
Кеічіп їетрегаіиге:3в3
```

РИС. 2.6. Результати роботи програми ех2_2. Обчислення температури за шкалою Кельвіна

2.3. Прості типи даних

Поняття типу даних є одним із фундаментальних понять програмування. Тип даних визначає:

+ *множину допустимих значень,* яких може набувати змінна або константа зазначеного типу;
• *множину допустимих операцій,* що застосовуються до даних певного типу;
+ спосіб зображення даних у пам'яті комп'ютера.

Якщо типи даних у мові PascaI класифікувати за будовою та властивостями відповідних даних, то можна виділити такі типи: прості, структуровані, посилальні, процедурні й об'єктні. Дані *простих типів* не містять у собі як складові елементи дані інших типів. Натомість дані *структурованих типів* - це об'єднання певної кількості елементів даних. Процедурні типи розглядатимуться у розділі 4, об'єктні - у розділі 6, а посилальні — у розділі 10.

З іншого боку, за способом утворення розрізняють *стандартні типи даних* та *типи даних користувача.* Тип даних користувача створюється програмістом і діє в межах певної програми чи модуля, а стандартний тип даних є елементом мови програмування.

Ще один спосіб класифікації дає можливість виділити *порядкові типи даних.* Множина допустимих значень порядкового типу являє собою упорядковану послідовність, кожний елемент якої має свій порядковий номер. Детальніше властивості порядкових типів даних розглядатимуться у розділі 2.3.8.

Під час збереження значень певного типу даних використовується певний обсяг оперативної пам'яті, який можна визначити вбудованою функцією зі geoT. Аргументом цієї функції може бути ідентифікатор типу даних, змінна або типізована константа.

У розділі 2.3.1 буде означено поняття операції над даними, а решту розділу 2.3 присвятимо розгляду простих типів даних. До простих типів даних належать цілі типи, дійсні типи, символьний тип, булів тип, перелічуваний тип та інтервальний тип. Усі прості типи, окрім дійсних, є порядковими. Цілі типи, дійсні типи, символьний тип та булів тип є стандартними, а перелічуваний та інтервальний — типами користувача.

2.3.1. Операції над даними

Операції описують дії, які необхідно виконати над певними значеннями, що можуть бути значеннями констант, змінних, функцій або виразів. Значення, до якого застосовується операція, називається її *операндом.* Операція позначається спеціальною лексемою — *символом операції.* Прикладами символів операції можуть бути лексеми +, a nci, [] тощо.

Залежно від типів операндів операції поділяються на арифметичні, логічні, рядкові тощо. Наприклад, арифметичні операції застосовуються до значень числових типів даних, а логічні операції — до логічних значень.

Залежно від кількості операндів операції в мові РазсаІ поділяються на унарні та бінарні. *Унарні* операції застосовуються до одного операнда. Символ унарної операції записується перед операндом, наприклад: *-а. Бінарні* операції застосовуються до двох операндів; символи таких операцій записуються між операндами, наприклад: *х + у.*

2.3.2. Цілочислові типи

Цічочислові типи — це типи даних, множини допустимих значень яких є множинами цілих чисел. Ідентифікатори цілочислових типів у мові РазсаІ, множини допустимих значень цих типів та обсяги пам'яті, що потрібні для збереження відповідних даних, наведені у табл. 2.1.

Таблиця 2.1. Цілочислові типи

Ідентифікатор типу	Кількість байтів оперативної пам'яті	Діапазон значень
оуїе	1	0 .. 255 (2^8 - 1)
зііогїі іиЬ	1	-128 .. 127
"пїедег	2	-32 768 .. 32 767 (2^{15} - 1)
мгй	2	0 .. 65 535
Іопдіпі	4	-2 147 483 648 .. 2 147 483 647 (2^{31} - 1)

Над усіма цілочисловими типами означений однаковий набір операцій. Усі ці операції, крім однієї, є бінарними. Символ «-» є символом як бінарної операції віднімання, так і унарної операції «мінус»: -32 768, -(2 - 3). Арифметичні операції над даними цілочислових типів перелічено у табл. 2.2 разом із прикладами їх застосування. Значення цілочислових типів записують у десятковій або шістнадцятковій системі числення. Ціле число не може містити десяткової точки. Для запису значення у шістнадцятковій системі перед символами числа записують символ «$», наприклад, $PPA1, що дорівнює десятковому значенню 65 441.

Таблиця 2.2. Арифметичні операції над цілочисловими значеннями

Знак операції	Зміст операції	Приклади застосування та результати
	Додавання	1 + 2 = 3; 1 + 32 767 - -32 768 (переповнення комірки); -32 7(38 + (-32 768) = 0
	Віднімання	1 - 2 = -1.; -32 768 - 1 = 32 767; 32 768 - (-32 768) = 0
- (унарний)	Зміна знаку числа	-(1) = -і; -(-32 768) = -32 768 (переповнення комірки)
	Множення	2 * 2 = 4; 256 * 128 = -32 768 (переповнення комірки); 256 * 256 = 0; 32 767 * 32 767 = 1
зі V	Визначення цілої частини від ділення	7 СПУ 3 = 2;-7 сіі V 3 = -2; 7 СПУ -3 = -2; -7 СПУ -3 = 2

продовження #

Таблиця 2.2 *(продовження)*

Знак операції	Зміст операції	Приклади застосування та результати
ПКЭП	Визначення остачі ВДД ділення	7 тосі 3 = 1; -7 тосі 3 = -1; 7 тосі -3 = 1; -7 тосі -3 = -1
/	Ділення	7/3 = 2.333 333 3 (дійсне), б/3 = 2.0 (дійсне)

Пояснимо зміст операцій сшу та ппос). Операція цілочислового ділення сшу відкидає дробову частину частки. Отже, результатом операції а сшу Ь є число [*a/Ь],* тобто найбільше ціле число, що не перевищує *a/Ь.* Операція тосі (ділення за модулем) визначає залишок від ділення двох чисел. Таким чином, за будь-яких значень а та Ь виконується рівність а сііV Ь + а тосі Ь = а.

Значення типів зЬогІіпі:, іпіедег та Іопдіпї є знаковими, а типів Ьуіе та моrсі - беззнаковими. При додаванні, відніманні та множенні знакових цілих чисел можливе перенесення одиниці зі старшого цифрового розряду в знаковий розряд. Така ситуація називається *переповненням.* Саме це відбувається при додаванні, наприклад, 1 і 32 767. Оскільки усі значущі розряди числа 32 767 дорівнюють 1, то число 32 768 = 32 767 + 1 має одиницю у знаковому розряді, а в решті розрядів - нуль. Це означає, що число стає від'ємним і інтерпретується як -32 768. При відніманні одиниці від -32 768 одиниця у знаковому розряді перетворюється на нуль, а отже результатом є додатне значення 32 767. Одиниця, яка переноситься із знакового розряду вліво за розрядну сітку комірки, втрачається. Тому, наприклад, при подвоєнні значення -32 768 отримуємо 0.

УВАГА ——

Переповнення комірки оперативної пам'яті під час виконання арифметичних операцій над цілими значеннями не вважається помилкою. Наслідком цього є «зацикленість» цілочислових типів у тому розумінні, що додавання одиниці до найбільшого числа дає найменше число. При переповненні комірки оперативної пам'яті одержується результат у межах множини значень цілочислового типу, але не обов'язково правильний.

Зауважимо також, що під час виконання операції ділення дільник не може дорівнювати нулю. У протилежному випадку станеться аварійне завершення програми з виведенням повідомлення: Еггог 200: ОІУІЗІОП Ьу гего (Ділення на нуль). Застосування операції ділення до цілих чисел дає результат дійсного типу.

Розглянемо операції *порівняння* цілих чисел (операції *відношення*). Ці операції позначаються лексемами =, <>, >, <, >=, <= («дорівнює», «не дорівнює», «більше», «менше», «не менше», «не більше»). Операції відношення визначають, чи є істинним значення виразу порівняння. Таким чином, результат операції порівняння матиме логічний тип. Наприклад, порівняння 1 = 2 дає результат Таізе, а порівняння 1 <> 2 і 1 >= 1 - результат їгие тощо.

2.3.3. Дійсні ТИПИ

Множина допустимих значень будь-якого дійсного типу є скінченною підмножиною множини раціональних чисел і містить, зокрема, усі цілі числа типу іпіедег.

Для запису дійсних чисел в оперативній пам'яті використовується розглянута в розділі 1 *форма з плаваючою комою.* Дійсне число у формі з плаваючою комою має мантису та порядок. Кількість цифр в мантисі характеризує точність числа. Чим більше цифр у мантисі, тим вище точність. Порядок визначає справжнє місцезнаходження десяткової точки в числі.

У мові РазсаІ означено п'ять дійсних типів (табл. 2.3): дійсний (reai), дійсний з одинарною точністю ($inдіє), дійсний з подвійною точністю (cioиble), дійсний з підвищеною точністю (exiencieci), цілий у форматі дійсного типу (сотр). Серед усіх дійсних типів exiencieci має найширший діапазон і найвищу точність, але потребує при цьому найбільших витрат пам'яті.

Таблиця 2.3. Дійсні типи даних

Назва	Кількість байтів оперативної пам'яті	Найменше за модулем число	Найбільше за модулем число
зшдіе	4	1,5-10 ^{45}	3,4-Ю^{38}
reai	6	2,9-Ю"^{39}	1,7- 10^{за}
cioиble	8	5-Ю"^{324}	1,7-Ю^{308}
exiencieci	10	3,4-10$^{\sim 4932}$	і,1-Ю^{4932}
сотр	8	-2^{63}+1ж-9,2-10^{18}	2^{63}~1~9,2-10^{18}

Дії над даними типів зіпдіе, cioиble, exiencieci і сотр виконуються тільки за наявності математичного співпроцесора. Його застосування значно підвищує точність розрахунків і прискорює їх виконання. РазсаІ надає можливість емулювати роботу математичного співпроцесора програмним способом за допомогою директив компілятора {$N-l-. E-}.

Тип сотр (від сотроипсі — складений) містить 64-бітні цілі числа, але у дійсному форматі. У виразах цей тип є сумісним із дійсними та цілими типами, але застосовувати до даних типу сотр можна лише визначені для дійсних типів даних операції.

Запис дійсного числа містить обов'язкову *цілу частину,* за якою вказуються *дробова частина* і *порядок.* Ціла частина - це непорожня послідовність цифр, дробова — непорожня послідовність цифр із крапкою на початку, а порядок - це дві або чотири цифри зі знаком «+» або «-». Цифри порядку записані після літери «Е» або «е». Перед значенням від'ємного дійсного числа записується знак «-», наприклад: -1.2345678900E-01. Дійсне число, в якому перед десятковою точкою записано цифру від 1 до 9, називається *нормалізованим.*

Для дійсних типів означено чотири арифметичні операції: додавання (+), віднімання (-), множення (*), ділення (/). Також над даними дійсних типів можна виконувати ті самі операції порівняння, що і над даними цілих типів. Зауважимо, що у загальному випадку дійсні числа можна порівнювати лише наближено, а застосування операції порівняння (=) до рівних з математичної точки зору виразів може дати результат ґаїзе.

2.3.4. Булів тип

Множина допустимих значень булевого, або логічного, типу містить дві константи: Таїзе (хибність) і ігие (істина). Назва цього типу даних походить від прізвища видатного англійського математика Джорджа Буля, засновника математичної логіки. Ідентифікатором логічного типу є слово Ьооїеап.

До булевих значень застосовуються операції «і», «або», «не», що називаються відповідно *логічніш множенням (кон'юнкцією), логічніш додаванням (диз'юнкцією)* й *запереченням* і позначаються лексемами апсІ, ог та поі. Ще одна операція називається «виключне або» чи «додавання за модулем 2» і позначається лексемою **хог.** Результати застосування цих операцій до булевих значень наведено в табл. 2.4.

Таблиця 2.4. Булеві операції

А	в	А апсі В	А ог В	А хог В	поі А
Таїзе	Таїзе	Таїзе	Таїзе	Таїзе	Ігие
Таїзе	ігие	Таїзе	ігие	Тгие	Тгие
їше	Таїзе	Таїзе	їгие	Тгие	Таїзе
їгие	ігие	їгие	їгие	Таїзе	Таїзе

Зазначимо, що булеві операції поі, апсі, ог і хог можуть бути застосовані не лише до логічних значень, а й до цілих чисел. При цьому булева операція застосовується до окремих розрядів або пар розрядів у двійковому записі операндів за такими правилами:

```
1 апсі 1 - 1;  | 1 апсі 0 = 0;  | 0 апсі 1 = 0;  0 апсі 0 = 0;
1 ог   1 - і;  | 1 ог   0 = 1;  | 0 ог   1 = 1;  0 ог   0 = 0;
1 хог  1 = 0;  | 1 хог  0 - 1;  | 0 хог  1 = 1;  0 хог  0 = 0.
```

Для прикладу розглянемо обчислення виразу б апсі 3. У двійковій системі числа 6 та 3 мають вигляд 110 і 011 відповідно. До кожної пари розрядів цих операндів застосовується операція апсі: 110_2 апсі $011_2 = 010_2$. Двійковий результат 010 дорівнює десятковому числу 2.

2.3.5. Символьний тип

Множина допустимих значень символьного (літерного) типу — це множина символів кодової таблиці А5СІІ, а отже, даними цього типу є окремі символи. Кожному символу відповідає ціле число (код) в діапазоні від 0 до 255. Зберігання одного символу потребує одного байта оперативної пам'яті. Нагадаємо, що символи з кодами від 0 до 127 відповідають стандарту А5СІІ. Вони ідентичні на всіх IBM-сумісних комп'ютерах. Символи з кодами від 128 до 255 можуть варіюватися залежно від типу комп'ютера та установок системного програмного забезпечення. Символьний тип позначається ідентифікатором сїіаг.

Значення символьного типу даних записуються в одинарних лапках. Наприклад, 'А', '1', '+' тощо. Сам символ одинарних лапок задається подвоєнням свого

значення: '1''. Будь-яке символьне значення можна задати також за допомогою стандартної функції сііг(і), де і - вираз цілого типу зі значенням від 0 до 255, тобто код символу. Функція повертає значення символу за його кодом. Наприклад, спг(48) - це символ '0', код якого дорівнює 48, сііг(49) - це символ 'I', сіг(65)— це символ 'A', а сИг(97) - символ 'a'. Також значення символу можна отримати за його АЗСІІ-кодом. Для цього перед значенням АЗСІІ-коду записується префікс #, наприклад: #48, #65, #97. Символи з кодами від 0 до 31 належать до керуючих символів. Якщо ці коди використовувати у процедурах введення-виведення, то можна управляти розташуванням даних на екрані, супроводжувати звуковими сигналами певні дії тощо. Керуючі символи можна також отримати записом у програмі комбінації символа «^» і латинської літери (табл. 2.5).

Таблиця 2.5. Деякі керуючі символи таблиці АЗСІІ кодів

Код	Назва символу	**Спосіб** введення	**Запис у програмі**
7	Звуковий сигнал	сні+6	
9	Горизонтальна табуляція	ан+і	"I
10	Переведення рядка	СПІчО	
11	Вертикальна табуляція	агі+і.	"I
13	Повернення каретки	СіхІ+М	"Н

Для перетворення маленьких літер на великі використовують функцію Іірсазе, синтаксис виклику якої є таким:

,ірса$е(<символ>)

Тут <символ> - маленька латинська літера. Функція Іірсазе не обробляє кирилицю.

Для значень символьного типу означено операції порівняння. Символи вважають рівними, якщо рівні їх АЗСІТкоди. Один символ вважають більшим за інший, якщо його А5СІІ-**КОД** більший. Зокрема,

'0' < '1' < ... < '9' < 'A' < 'B' < ... < 'V < 'a' < 'із' < ... < '2'.

Крім операцій порівняння для даних символьного типу означено операцію *конкатенації (об'єднання*), у результаті виконання якої утворюється рядок. Цю операцію позначають символом «+». Наприклад: ' 1'+'2' = '12'.

2.3.6. Перелічуваний тип

Перелічуваний тип означується користувачем. Такий тип задається переліком усіх елементів множини допустимих значень. Кожне значення іменується певним ідентифікатором і зазначається у списку, який береться у круглі дужки. Ідентифікатор перелічуваного типу треба оголосити у програмі в розділі Іуре. Синтаксис оголошення перелічуваного типу такий:

їуре «ідентифікатор типу> = (<ідентифікатор_1>.<ідентифікатор_2>.....<ідентифікатор_п>);

Тут <ідентифікатор типу> - це ідентифікатор перелічуваного типу; <ідентифікатор_1>....<ідентифікатор_п> - допустимі значення перелічуваного типу. Наприклад:

```
type IlIogkMeek = (Моп. Тие. Mecl. ТИи. Pгі, 5a"Ь, 5ип);
     Соїог = (гесі.дгееп.ЬИие);
     кIіпїегМопїН = ШесетЬег,Запиагу.РеЬгиагу);
```

Елементи перелічуваного типу впорядковані за номером елемента в оголошенні типу. При цьому перший ідентифікатор у списку отримує порядковий номер 0, другий ідентифікатор - номер 1 і т. д. Максимальна кількість ідентифікаторів в оголошенні перелічуваного типу становить 65 536. Тому перелічуваний тип можна розглядати як підмножину цілочислового типу mogsi. Для перелічуваних типів означені операції порівняння =, <>, <, >, <=, >=, що порівнюють порядкові номери своїх операндів.

Під час використання перелічуваних типів слід враховувати, що PascaI не підтримує операцій введення та виведення значень перелічуваного типу.

2.3.7. Інтервальний тип

Інтервальний тип, як і перелічуваний тип, означується користувачем. Цей тип задається діапазоном значень базового типу, роль якого може виконувати будь-який інший порядковий тип. В оголошенні інтервального типу вказують мінімальне та максимальне значення діапазону, розділяючи їх лексемою «..». Наведемо синтаксис оголошення інтервального типу:

```
type ідентифікатор типу> =
«Мінімальне значення> .. «Максимальне значенням
```

Нижче наведено приклад такого оголошення:

```
type сИдіI = 0..9: ЬаПіп = 'А'.,'Г \ МогкИеек = Моп..Pгі;
```

При оголошенні інтервального типу слід дотримуватися таких правил:

- два символи «..» розглядаються як лексема, тому пробілів між ними бути не може;
- обидві константи, які визначають межі діапазону, мають належати тому самому базовому типу;
- значення лівої межі діапазону має бути меншим від значення правої межі.

2.3.8. Порядкові типи

Як уже було зазначено вище, множина допустимих значень порядкового типу даних є упорядкованою послідовністю, кожний елемент якої має свій номер. До порядкових типів даних належать цілочислові типи, символьний, логічний, перелічуваний та інтервальний тип. Дійсні типи даних не є порядковими, оскільки множина допустимих значень будь-якого дійсного типу, крім типу зі пді е, є надто великою для того, щоб її елементи можна було пронумерувати значеннями певного цілочислового типу.

Для обробки даних порядкових типів визначені такі стандартні функції:

- функція orci повертає порядковий номер свого аргументу;
- функція prec повертає значення, що передує вказаному;
- функція зисс повертає значення, що є наступним після значення аргументу функції зисс.

Розглянемо особливості застосування цих функцій до даних символьного, булевого та інтервального типу.

Якщо x є символом, то функція orci(x) повертає його АЗСП-код, наприклад: orci('O') = 48, orci('A') = 65, orci('a') = 97. Як уже згадувалося в розділі 2.3.5, для символьного типу означена також функція сіїг, що обчислює символ за його кодом. Функції сЬг і orci є зворотними, тобто cHr(orcKx)) = x для будь-якого символу x, а orcKcIir(n)) = n для будь-якого цілого числа n від 0 до 255.

Для значень булевого типу в мові РаscaI означено співвідношення orcICTaïze) = 0 та orcKirue) = 1. Такому порядку булевих значень відповідає результат їх порівняння: Таїze < "irue.

Якщо функцію orci застосувати до значення інтервального типу, то вона поверне порядковий номер відповідного елемента базового типу.

УВАГА

Усі порядкові типи «зациклено» так, що наступним за елементом із найбільшим номером є елемент з найменшим номером, а попереднім перед елементом із найменшим номером є елемент із найбільшим номером.

Отже, якщо функцію preci застосувати до першого допустимого значення порядкового типу, вона поверне значення з останнім порядковим номером. Наприклад, якщо значення змінної x типу Ьуїе дорівнює нулю, то preci(x) = 255. Аналогічно, якщо функцію зисс застосувати до останнього допустимого значення, вона поверне значення з найменшим порядковим номером. Наприклад, якщо значення змінної x типу т г б дорівнює 65 535, то зисс(x) = 0.

2.4. Константи, змінні, вирази

Будь-які значення, що використовуються у програмі, - це або значення змінних, або константи. Принципова відмінність між змінними і константами полягає у тому, що для зберігання значень змінних під час виконання програми відводяться ділянки пам'яті, а константи є частиною коду програми. Тому в процесі виконання програми значення змінної може модифікуватися, а константа — ні. Використовуючи константи, змінні та операції, можна утворити вирази, що застосовуються для опису обчислень.

2.4.1. Різновиди констант

Константа в тексті програми РаscaI може позначатися лексемами двох типів:

- безпосередній запис значення;
- ідентифікатор.

У першому випадку константа може бути, наприклад, послідовністю цифр (цілим числом), послідовністю символів в одинарних лапках (рядком) тощо.

У другому випадку певному значенню приписується ідентифікатор. Це відбувається в розділі оголошення констант. Наприклад, наближене значення величини V2 можна позначити ідентифікатором здг1:2:

```
c0n5ï
  $ЯГЇ2=1.4l42;
```

За умови такого оголошення вирази здг£2 та 1.4142 завжди матимуть однакове значення. Зручність позначення констант ідентифікаторами виявляється тоді, коли одна й та сама константа згадується в тексті програми багато разів. Завдяки позначенню ідентифікатором корекція такої константи не потребуватиме її пошуку в усьому тексті програми, а може бути здійснена шляхом корекції значення в оголошенні ідентифікатора.

Як і будь-які інші значення, константи належать певним типам даних.

Числові константи — це цілі десяткові, цілі шістнадцяткові та дійсні десяткові числа. Цілі десяткові числа записують стандартним способом. Вони мають знаходитися в діапазоні від -2 147 483 648 до 2 147 483 647, наприклад: 123, -1024. Перед шістнадцятковим цілим числом записують символ «$». Нагадаємо, що для зображення шістнадцяткового числа використовуються цифри від 0 до 9, а також латинські літери від A (a) до P (i), наприклад: $7B, $400. Допустимий діапазон шістнадцяткових цілих чисел — від $00 00 00 00 до $PP PP PP PP. Дійсні числа записуються у вигляді звичайного десяткового дробу або в експоненціальній формі з основою 10. У десяткових дробах ціла частина від дробової відокремлюється крапкою, наприклад: 3.1415926, -27.18283. В експоненціальній формі запису дійсного числа замість основи 10 використовується буква «E», після якої ставиться показник степеня: 3.141526E+00, -2.718283E+01.

Символьна, константа — це АЗСІІ-символ, записаний в одинарних лапках (апострофах), або символ, записаний за допомогою АЗСІІ-коду, перед яким вказано префікс «#», наприклад: 'A', #13.

Рядкова константа є послідовністю символів, що записуються в одинарних лапках (апострофах), наприклад: 'Вогіапсі Pascal 7.0'. Якщо рядкова константа містить апостроф, то він записується двічі поспіль: 'Комп' 'ютерні науки'.

Логічні константи мають значення Раї 5Є або ігиє, що означає хибність та істинність відповідно.

У прикладі 2.3 наведені оголошення іменованих констант у PascaI.

Приклад 2.3

```
conzï
     cИдії = 1000;                        {цілочислова константа}
     KeaIConzï = 3.14;                    {дійсна константа     }
     ЗутЬoIConzi =    ;                   {символьна константа  }
     EзcKey = #27;                        {символьна константа  }
     ЗїгіпдСопзГ = 'зесїіоп оР сопзіапї': {рядкова константа    }
     BooIConzï = Раїзе:                   {булева константа     }
```

Одна з особливостей констант полягає в тому, що компілятор не виділяє оперативної пам'яті для їх зберігання; він автоматично розпізнає тип константи без його попереднього оголошення. Під час компіляції програми компілятор замінює ідентифікатори констант їх значеннями в операторах, де ці ідентифікатори трапляються. Ідентифікатори констант припиняють своє існування після компіляції. Якщо користувач хоче зберегти значення констант і виділити для них оперативну пам'ять, він може використати типізовані константи.

Типізовані константи — це змінні, яким надано значення на початку виконання програми. Такі константи називаються змінними, що ініціалізуються, або змінними з початковим значенням. Синтаксис оголошення типізованої константи;

```
сопзї <ідентифікатор>:<тип> = <вираз>;
```

Тут значення вказаного справа від знака «=» виразу повинне мати той самий тип, що й ідентифікатор константи.

Наведемо приклад оголошення типізованих констант.

Приклад 2.4

```
сопзі;
     year:іпіедег  = 2004;
     ргice;reai    = 250.56;
     апзмег:cHar   = 'y';
     Ьei 1 : ciiar    = #7;
     T1ад;Ьoo1eап  = Таїзе;
```

Оскільки типізовані константи фактично є змінними певного типу, вони можуть набувати нових значень під час виконання програми, але не можуть згадуватися в оголошеннях інших констант або типів.

Зазначимо ще одну властивість оголошених у процедурах і функціях (див. розділ 4) типізованих констант. Такі константи не ініціалізуються знову під час повторного виклику процедури (або функції), вони зберігають значення, отримані під час її попереднього виклику.

2.4.2. Змінні

Змінна величина — це узагальнення, абстракція якогось реального чи уявного об'єкта, що може перебувати в різних станах. Змінна може характеризувати окремі властивості об'єкта, що змінюються під впливом зовнішніх умов. Зазвичай вона позначається *ідентифікатором (ім'ям);* так у другому законі Ньютона, що може бути відображений формулою $a = P/m$, імена m, a, P позначають змінні величини - масу тіла, прискорення його руху та силу, що діє на тіло.

Змінні у програмуванні призначені для зберігання та передачі даних усередині програми. На відміну від констант, які не можуть змінювати свої значення, змінні набувають різних значень під час виконання програми.

Змінна має свій ідентифікатор і належить до певного типу. Нагадаємо, що бажано використовувати мнемонічні ідентифікатори, пов'язані з певними поняттями.

Тип змінної задає множину її допустимих значень, множину операцій, які можна застосувати до неї, а також необхідний для збереження значень змінної обсяг оперативної пам'яті.

Отже, змінні виконують функцію зберігання даних. З іншого боку, відомо, що під час виконання програми дані зберігаються в комірках оперативної пам'яті комп'ютера. Тому на фізичному рівні поняттю змінної відповідає група комірок оперативної пам'яті. Кожна така комірка має свою *адресу.* Оскільки програмісту не досить зручно працювати з адресами оперативної пам'яті, то ці адреси ставляться у відповідність ідентифікаторам змінних під час їх оголошення. Таким чином, ім'я змінної *вказує,* або *посилається,* на групу комірок оперативної пам'яті (рис. 2.7). Якщо говорити більш точно, ім'я змінної посилається на першу комірку з групи, а величину групи визначає тип змінної.

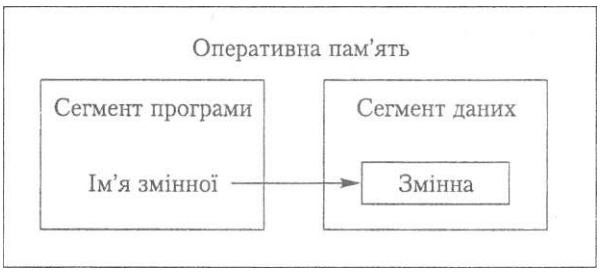


Рис. 2.7. Ідентифікатор змінної та її асоціація з коміркою пам'яті

У мові PascaI змінні оголошуються в розділі оголошення змінних, що відкривається ключовим словом уаг (від англ. уагіаЬIe - змінна). Нагадаємо синтаксис оголошення змінної:

уаг <і дентифі катор>:<тип>;

Зазначимо, що однотипні змінні можна оголошувати в одному переліку, вказуючи їх імена через кому. Як ідентифікатор типу можна використовувати ім'я, яке було оголошене раніше у розділі Туре, або ім'я стандартного типу. У прикладі 2.5 наведено оголошення змінних різних типів.

Приклад 2.5

```
їуре
ИогкЬIеек = (Моп, Тие, Месі. Тіш, Ргі, 5аТ, 5ип);
  кІпїегМопТЬ = ШесешЬег,Оапиагу.РеЬгиагу);
маг
  а,Ь,с:геаі:              {дійсні змінні              }
  і,з:іпТедег;             {цілочислові змінні         }
  Тад:Ьоо1еап:             {булева змінна              }
  кеу:сИаг;                {символьна змінна           }
  сНдії:0.,9;              {інтервальний тип користувача }
  аірИаЬеО:'А'..'2':       {інтервальний тип користувача }
  сіау:иогкУеек;           {перелічуваний тип користувача}
  СОІСШМІПШЄГМОПІН;        {перелічуваний тип користувача}
```

Змінні можна оголосити в різних частинах програми:

- в розділі уаг оголошення змінних програми;
- в інтерфейсній частині модуля, який потрібно підключити до програми за допомогою зарезервованого слова изез;
- у процедурах і функціях.

Залежно від того, де оголошена змінна, ЇЙ надається *область видимості,* тобто область, де її можна використовувати. Докладніше області видимості змінних розглядатимуться в розділі 4.

Зазначені вище оголошення покладають на компілятор функцію встановлення відповідності між іменем змінної та адресою пам'яті. Але мова РазсаІ надає розробнику можливість встановити таку відповідність самостійно. Йдеться про *абсолютні* змінні. Є дві форми оголошення абсолютних змінних: із зазначенням точної адреси змінної та із розташуванням двох змінних за однією адресою.

Перша форма оголошення абсолютних змінних вимагає вказати ідентифікатор змінної, її тип, ключове слово аЬзоІиіе і повну адресу (сегмент і зсув), за якою буде записане значення змінної:

```
уаг <ідентифікатор>:<тип> аЬзоІиїе $<сегмент>:$<зсув>;
```

Тут $<сегмент> - номер сегмента даних, а $<зсув> - значення зсуву від початку сегмента даних в байтах. Ці шістнадцяткові константи не повинні виходити за межі діапазону від $0000 до $PPPP. Наприклад:

```
уаг уа!ие:Ьу1:е аЬзоІиїе $0040:$0049:
```

Інша форма оголошення абсолютних змінних вимагає вказати ідентифікатор змінної, її тип, ключове слово аЬзої и£е та ідентифікатор іншої змінної або типізованої константи:

```
уаг <ідентифікатор_1>:<тип> аЬзоІиіе <ідентифікатор_2>;
```

Тут <і дентифі катор_1> - ідентифікатор абсолютної змінної; <ідентифікатор_2> — ідентифікатор іншої змінної, за адресою якої буде розташована абсолютна змінна. Наприклад:

```
уаг
  сІідП:0. .9;
  аЬзуагИгНедег аЬзоІиїе сіідії:
```

Ідентифікатори оголошених в одній області видимості змінних не повинні збігатися. У разі порушення цього правила компілятор видає повідомлення про помилку: **Етог 4: Рирісаіе ісіепйіїег** (Повторення ідентифікатора).

2.4.3. Вирази

Константи та змінні можна використовувати у *виразах.* Вираз є послідовністю операцій, операндами яких можуть бути змінні, константи, виклики функцій та інші вирази. Для керування порядком виконання операцій застосовуються круглі

дужки. У результаті послідовного виконання всіх операцій, що входять до складу виразу, обчислюється його значення.

Вираз може складатися лише з однієї лексеми, що позначає константу, змінну або виклик функції. Значення такого виразу має той самий тип, що і позначений лексемою елемент даних.

Якщо вираз без круглих дужок містить декілька операцій, то послідовність їх виконання визначається правилами *пріоритету*. У табл. 2.6 групи операцій мови Pascal розташовано у порядку зниження їх пріоритету. Найвищий пріоритет мають унарні операції, найнижчий — операції відношення. Операції всередині кожної групи мають однакові пріоритети.

Таблиця 2.6. Пріоритет операцій у мові Pascal

Групи операцій	Операції
Унарні	- (зміна знака), @, поі,
Мультиплікативні	*,/, апсі, тосі, сПУ
адитивні	+, -, ог, хог
Відношення	=, <>, <, >, <=, >=, іп

Отже, наведемо правила визначення пріоритету операцій:

- операнд, що міститься між двома операціями з різними пріоритетами, зв'язується з операцією, яка має вищий пріоритет;
+ операнд, що міститься між двома операціями з рівними пріоритетами, зв'язується з операцією, яка записана ліворуч;
- вираз, який взято в дужки, обчислюється в першу чергу і далі розглядається як окремий операнд;
- операції з однаковим пріоритетом виконуються зліва направо.

Розглянемо декілька прикладів застосування цих правил (табл. 2.7).

Таблиця 2.7. Застосування правил визначення пріоритету операцій

Вираз	Значення	Коментар
7/4*3	Ц = 5,25 4	Спочатку обчислюється 7/4=1.75, після цього — 1.75*3=5.25
3+7 тосі 4	6	Спочатку обчислюється 7 тосі 4=3, після цього — 3+3=6
N01 Таїзе ог Таїзе апсі їгие	Тгие	Ця операція рівнозначна такій: (пої Таїзе) ог (Таїзе апсі Тгие). Оскільки по{ Таїзе=1пие, то значення виразу становить Тгие незалежно від результату операції в других дужках
5>2 ог 7>8	Помилка	ог має пріоритет над відношенням. Тому спочатку буде виконано побітову операцію 2 ог 7=7 і отримано некоректний вираз 5>7>8

Коректний запис останнього виразу з табл. 2.7 буде таким: (5>2) ог (7>8).

Мова Pascal припускає оголошення константних виразів, що будуть обчислені під час компіляції програми, а не під час її виконання. Константні вирази слід за-

писувати за тими ж правилами, що і звичайні вирази, але в них припускається використання лише таких вбудованих функцій: аЬ§, спг, ocici, orci, prec, 5ucc, roupci, їгипс, 5I2Є0T. Декілька константних виразів оголошено у прикладі 2.6.

Приклад 2.5

```
conzT
     c!idiT=1000;                        {константні  вирази}
     Миїtieri caIExpreszsion = 976+453:
     3TripdExpreszsion = 'TurЬo ' + 'RazcaI':
     BooiExpreszsiun = 5 anci 3;
     KeTerExpreszsion = cIidiT < MitєricaiExpreszsion:
```

2.5. Найпростіші оператори

Оператори визначають дії, що мають здійснюватись комп'ютером під час виконання програми. Оператори мови RazcaI можна поділити на *прості* та *складені*. Прості оператори, на відміну від складених, не містять в собі інших операторів. До групи простих операторів належать оператори присвоєння та виклику процедур. Складені оператори розглядатимуться в розділі 3,

Програма — це послідовність операторів. Будь-які оператори відокремлюються один від одного символом крапки з комою (;). Крапка з комою не є частиною оператора, це роздільник. Відсутність крапки з комою між операторами спричиняє синтаксичну помилку Ertor 85: ";" exрейєсі (Очікується «;»).

2.5.1. Оператор присвоєння

Як уже зазначалося, змінні призначені для того, щоб описувати стани об'єктів, які моделюються у програмі. Надати значення змінній можна за допомогою оператора присвоєння або оператора виклику процедури введення значень.

Оператор присвоєння має такий вигляд:

<1дентифікатор> := <вираз>;

Тут ідентифікатор» - це ім'я змінної; знак присвоєння := — лексема, яку не слід плутати зі знаком операції порівняння (=).

За допомогою оператора присвоєння виконуються дії, які можна записати у вигляді такого алгоритму:

1. Обчислити значення виразу, записаного праворуч від символу присвоєння.
2. Надати обчислене значення змінній, позначеній ім'ям ліворуч від символу присвоєння.

Наприклад, якщо ім'я 2 оголошено як уаг 2 : і гтседег, то оператор присвоєння г:=7*(3+1) спочатку обчислює значення 28, а потім записує його до комірок пам'яті, що відповідають змінній 2. Після виконання оператора присвоєння змінна 2

має значення 28. Отже, оператор присвоєння дозволяє змінити поточне значення змінної.

Якщо у деякому виразі використовується змінна, яка ще не отримала значення, воно вважається невизначеним і позначається для числових типів нулем (для символьного типу — символом з ASCII-КОДОМ 0, для логічного типу — значенням Таїзе). У разі використання неоголошеної змінної, або змінної, що не належить поточній області видимості, компілятором буде видано помилку **Error 3: Identifier expected** (Невідомий ідентифікатор).

В ГОЕ Borland Pascal 7.0 можна здійснити покрокове виконання програми (див. підрозділ 2.1), використовуючи значення змінних, наведених у вікні **Watches**. У табл. 2.8 розглянуто покрокове виконання програми з прикладу 2.7.

Приклад 2.7

У програмі ex2_3 виконується серія присвоювань. Розглянемо процес модифікації значень змінних під час виконання цієї програми (табл. 2.8).

```
program ex2_3;
  var x, y, z : integer;
begin
  x := 1
  y := 3
  z := x + y;
  z := z + 10
end.
```

Таблиця 2.8. Покрокове виконання програми ex2_3

	Значення змінних (вміст комірок оперативної пам'яті)		
Оператор, що виконується	*X*	y	z
Початок програми	0	0	0
x := 1	1	0	0
y := 3	1	3	0
z := x + y	1	3	4
z := z + 10	1	3	14

2.5.2. Процедури введення даних

Важко навести приклад задачі, програмне розв'язання якої не містило б операцій введення-виведення даних. Введення даних — це процес їх передачі із зовнішніх носіїв інформації або пристроїв введення даних до комірок оперативної пам'яті для подальшої обробки. Виведення даних - це процес передачі їх з оперативної пам'яті на зовнішній носій інформації або пристрій виведення даних, яким може бути дисплей, принтер, магнітний диск тощо. Мови програмування, як правило, не мають операторів для виконання операцій введення-виведення. Ці операції здійснюються спеціальними процедурами введення та виведення.

Введення даних із зовнішніх пристроїв до оголошених у програмі змінних здійснюється за допомогою вбудованих *процедур введення* або *читання* (від англ. геасі - читати). Синтаксис процедур введення даних з клавіатури в мові РазсаІ такий:

```
геасі(<і дентифі катор_1>. <і дентифі катор__2>
      <ідентифікатор_п>);
геасЯп(<і дентифі катор_1>,<і дентифі катор_2>
        <ідентифікатор_п>);
```

Тут параметри процедури <і дентифі катор_1>. . . . .< і дентифі катор_п> — це імена змінних, яким будуть надані введені з клавіатури значення.

Процедури геасі і геасПп здійснюють введення символів, рядків і чисел. Це значить, що змінні, ідентифікатори яких є параметрами процедури читання, можуть бути будь-якого простого типу, крім булевого (цілого, дійсного, символьного), а також рядкового типу (див. розділ 7). Поряд зі змінними булевого типу як параметри процедури введення даних не дозволяється використовувати константи, оскільки їх значення не можуть бути змінені. Це правило не стосується типізованих констант, що є ініціалізованими змінними.

Коли викликається процедура геасі або геасПп для введення даних з клавіатури, виконання програми тимчасово переривається. В цей час користувач може вводити з клавіатури символи. Уведені символи запам'ятовуються у буфері та передаються процедурі введення тільки після натиснення клавіші **Enter.** Буфер — це область пам'яті для тимчасового зберігання даних. Максимальний обсяг буфера становить 128 символів (байтів). Завдяки наявності буфера можливе редагування даних під час їх введення. При редагуванні даних використовуються клавіші **Backspace, Ctrl+Z, Enter.** Введення символів супроводжується відображенням їх на екрані. Після натиснення клавіші **Enter** введений рядок символів перетворюється на значення того типу, який відповідає типу змінної у процедурі введення, а потім цій змінній присвоюється. Введені значення мають строго відповідати типам змінних у списку параметрів процедури. Якщо, наприклад, змінна має тип і піедег, а вводиться значення типу сііаг, то виникає помилка введення-виведення **Error 106:** Іт/аііісі **питегіс іогтаі** (Некоректний числовий формат) і робота програми припиняється. У разі, коли процедура введення має декілька параметрів, дані потрібно вводити послідовно через пробіл, а після останнього введеного значення необхідно натиснути клавішу Епіег. Якщо програма містить декілька викликів процедури -еасі поспіль, то дані для них можна набрати в один рядок, і тільки після цього натиснути клавішу **Enter.** Такий спосіб введення даних не придатний у разі використання процедури геасП п, особливості застосування якої будуть розглянуті нижче.

Приклад 2.8

```
уаг
  п:іпіедег;
  х:геаі:
  зушЬої:сНаг;
  з:зІгіпд:
Ьеділ
```

```
  reacKзутЬої);
  reacKп.х.з);
єпсі.
```

Під час виконання цієї програми значення даних можна набрати так:

```
А 567 3.I417 єпсі оТ іпзеПИоп  Епїег
```

У цьому прикладі спочатку вводиться значення символу, потім вводяться значення цілого та дійсного типу і рядок символів. Завершується введення натисненням клавіші **Enter.** Якщо спробувати ввести значення символу після числових значень, то виникне помилка введення-виведення **Error 106:** Inyaiісі питегіс іогтаі (Некоректний числовий формат). Таким чином, символьні значення в потоці даних вводити не бажано.

Під час введення числових значень процедура reacі, пропускаючи пробіли та символи табуляції на початку рядка, виділяє підрядок символів, що завершується пробілом, символом табуляції або символом кінця рядка. Підрядок, що виділено, розглядається як символьне зображення числової константи відповідного типу. Цей підрядок перетворюється на числове значення, яке надається певній змінній. Отже, числові значення, що вводяться із клавіатури, можна розділяти довільною кількістю пробілів і символів табуляції.

Процедура reacПп (від reacі Ііпе - читати рядок) застосовується аналогічно процедурі reacі. Але під час введення рядків або окремих символів рекомендується викликати саме процедуру reacїїп, яка зчитує всі символи рядка, включаючи символ завершення рядка. Символ завершення рядка формується під час натиснення клавіші Епіег. На відміну від процедури reacїїп, процедура reacі зчитує всі символи рядка, крім символу його завершення. Тому процедура reacі не забезпечує перехід до зчитування символів нового рядка.

Якщо викликати процедуру reacїїП без параметрів, програма чекатиме натиснення клавіші **Enter** для продовження своєї роботи. А якщо при потоковому введенні буде введено даних менше, ніж задано змінних у відповідному списку введення, процедура читання чекатиме продовження введення.

2.5.3. Процедури виведення даних

Виведення, або *запис* (від англ. **УГПЄ** - писати), значення виразу на дисплей здійснюється за допомогою процедури mгііеіп або mгііе за таким синтаксисом:

```
мгі£е1п(<список  виведення»):
мгі1:е(<список  виведення»):
```

Тут <список виведення» — це список записаних через кому виразів.

Процедури мгіїеіп і мгіїе виводять дані на екран у символьному вигляді. Виведення здійснюється зліва направо та згори донизу. Якщо рядок символів, що виводиться, має довжину, яка перевищує ширину вікна програми, то символи переносяться на новий рядок. Якщо виведення здійснюється в останньому рядку вікна програми, то вже відображені рядки символів зсуваються догори, у резуль-

таті чого верхній рядок зникає, а знизу додається порожній рядок. Такий спосіб відображення даних називається *прокруткою.*

Параметрами процедур игііеіп і мгііе можуть бути рядкові константи, що записуються в одинарних лапках. За допомогою рядкових констант можна виводити повідомлення:

```
мгіІе1п('Оешопзїгаїіоп сопзїапі ехргеззіоп:[1]);
```

Для виведення значень іменованих числових констант використовують їх ідентифікатори. Значення констант перетворюються у набір символів, які виводяться на екран або інший пристрій:

```
сопзї
    сИдіі; = 1000;            {іменова цілочислова константа}
Ьеділ
    мгіі:е1п(5);              {цілочислове значення          }
    игііе"Іп( 'сііділ;=' .сИдіі); {значення іменованої константи}
єпсі.
```

Параметри процедур мгіїеіп і игііе можуть бути булевими константами. Лапки в такому разі не потрібні:

```
мгііеіп(їгие);
```

У процедурах виведення як параметри можна використовувати константні вирази, наприклад:

```
сопзі:
   ВіїмізеЕхргеззіоп = 5 апсі 3;
Ьеділ
   игіІеІп(ВіїмізеЕхргеззіоп);
єпсі.
```

Найчастіше процедури виведення застосовують для відображення значень змінних на екрані дисплея. Як параметри процедур використовуються ідентифікатори змінних і вирази:

```
уаг
   і Ипїедег:
   а:геа!:
Ьеділ
 мгіІе1п(а.і,а+і);    {вивести значення змінних і виразу}
 игі£е1п(а>і);        {вивести значення виразу відношення}
єпсі.
```

Під час виведення числових значень використовують формати виведення. Формат задає ширину поля виведення, тобто кількість символів у екранному зображенні результату. Для дійсних чисел крім ширини поля виведення зазначають кількість позицій після десяткової точки. Якщо до змінних дійсного типу не застосовується форматування, то їх значення виводяться в експоненціальній формі. Формат задається двома або однією цифрою після двох крапок, які записують після ідентифікатора змінної або константи, наприклад:

```
мп1;е1п((іідіі:10):  {вивести ціле число в поле завширшки 10 символів}
мгіІе1п(а:8:3);      {дійсне число вивести в поле завширшки 8 символів,
                      кількість символів у дробовій частині числа дорівнює трьом}
```

Процедура мгііеіп, на відміну від процедури мгіїе, після виведення значень згідно зі списком виведення переведе курсор до нового рядка. Виклик процедури виведення після попереднього виклику мгі Te! п відобразить дані в новому рядку, в той час як виведення після попереднього виклику мгііе здійснюватиметься в тому самому рядку, з поточної позиції курсору.

Процедуру мгіTeIп можна викликати без параметрів, що спричинить переведення курсору на новий рядок без виведення даних.

Приклад 2.9___

Розглянемо приклад програми, в якій викликаються процедури виведення для відображення на екрані різних значень у різних форматах. Результати роботи програми представлено на рис. 2.8.

```
ргодгат ех2_4;
сопзТ
  сііджТ = 1000;                          {цілочислова константа}
  КеаIСопзї = 3.14;                       {дійсна константа      }
  ЗутЬоIСопзТ =                           {символьна константа   }
  ЕзсКеу = #27;                           {символьна константа   }
  ЗїгіпдСопзТ. = 'зеciіоп оТ сопзіапі'; {рядкова константа}
  ВооIСопзТ = Таїзе;                      {булева константа
                                          {типізовані константи
  уеаг:іпїедег = 2004;
  рri ce:reai    = 250.56;
  апзмег;сііаг   = 'у':
  Ьеіі:сИаг     =#7;
                                          {константні вирази
  МишегicaIExprеззіоп = 976+453;
  ЗТгіпдExprеззіоп = 'ТигЬо ' + 'РазсаI';
  ВооiExprеззіоп = 5 апсі 3;
  КеТегExprеззіоп = сИдіМіитегicaI Exprеззіоп;
уаг
   i;іпТедег;                             {ціла змінна          }
   а;rеа1;                                {дійсна змінна        }
Ьеgіп
     мгііе1п('СопзТапТ ехргеззіоп сіешо');
     «ГІПеїпС'=============================•);
     мгіIе1п(5);
     мгіTe1п( 'сііджТ=' .сііджії);
     мгіTe1п('ЕзсКеу='.ЕзсКеу);
     мгіTe!п('ТурecI сопзТ уеаг=',уеаг);
     мгіTеїпС'СопТгоI зутЬої Ьe11='.Ьеіі);
     мгііеІпС 'МшпепсаIExprеззіоп 976+453='.МитегicaiExprеззіоп);
     мгіTe1п('ЗїгіпдExprеззіоп;''ТигЬо'' + " РазсаI'' = '.ЗТгіпдExprеззіоп);
     мгіTe!п('ВооIExprеззіоп; 5 апсі 3='.ВооIExprеззіоп);
     мгіїе1п( 'КеТегExprеззіоп; сіі діП^Иитегі са! Exprеззіоп=' .КеТегExprеззіоп);
     мгіTe1п('Епіег іпТедег. геаі');
     геасіїп{i ,a);
     мгіTe1п('Exprеззіоп міТії уагіаЬIез сіето'):
     МПїеіП('=====================================•);
```

```
    «гііеІпС'Ехропепііаі Топш: a+i-',a+i):
    мпіеІпС'Рогтаііесі оиірии: a+i-' ,a:3:3,'+' .i. '=' .a+i :8:3):
    мгііе1п(a>i):
    мгіТеІп(Ігие);
    геасіїп;                {чекати натиснення клавіші Епіег}
єпсі.
```

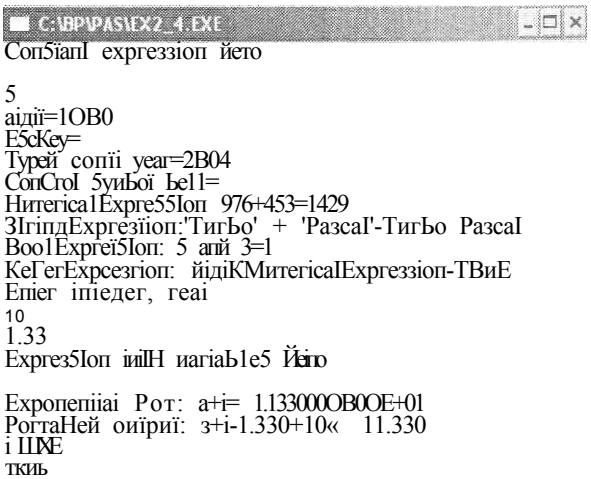


РИС. 2.8. Результати роботи програми ex2_4. Виконання процедур введення-виведення

2.5.4. Сумісність типів

Мова РазсаI є строго типізованою мовою програмування. У ній строго дотримується концепція типів даних, відповідно до якої всі операції, що застосовуються у мові, означені лише над операндами сумісних типів. Сумісність типів важливо враховувати у виразах та в операторі присвоєння значень змінним.

Можливість використання операндів різних типів у виразах називається *сумісністю типів.* Можливість присвоювати значення одного типу змінним іншого типу називається *сумісністю за присвоєнням.*

Два типи вважаються сумісними у виразах, якщо виконується хоча б одна з наступних умов:

+ типи двох операндів однакові;
- типи двох операндів дійсні;
- типи двох операндів цілочислові;
- тип одного операнда є піддіапазоном типу іншого операнда;
- типи двох операндів є піддіапазонами того самого базового типу;
- тип одного операнда рядковий, тип іншого рядковий або символьний;

- тип одного операнда — нетипізований покажчик, тип іншого операнда — будь-який типізований покажчик;
- типи двох операндів процедурні — вони означують процедури (функції) з однаковою кількістю параметрів, однаковими типами відповідних параметрів і однаковими типами значень, що повертаються (для функцій).

Розглянемо декілька прикладів сумісності у виразах. В одному виразі можна використовувати дійсні та цілочислові операнди (1+2.5). Під час обчислення такого виразу спочатку цілочисловий операнд перетворюється на дійсний, а потім виконується зазначена операція над дійсними значеннями. Операнди можуть мати різні цілочислові типи. Якщо один операнд знаковий, а другий беззнаковий, обидва перетворюються до цілочислового знакового типу, якому можуть належати значення обох операндів і який вимагає найменшого обсягу оперативної пам'яті. Наприклад, операнди типів зііогііпі і ЬуТе перетворюються на операнди типу 1 пРедег, операнди типів зіюгііпі: і иогсі - на операнди типу Іопдіпі. Якщо обидва операнди мають знакові цілочислові типи, або обидва мають беззнакові цілочислові типи, то операнд типу, що потребує меншого обсягу оперативної пам'яті, перетворюється на операнд типу, що вимагає більшого обсягу оперативної пам'яті (наприклад, операнд типу зНогііпі перетворюється на операнд типу іпіедег, операнд типу іпіедег — на операнд типу ІопдіпТ, а операнд типу ЬуТе — на операнд типу моrсі).

Під час виконання оператора присвоєння значення змінній обов'язково потрібно враховувати сумісність типів. Тип значення виразу є сумісним за присвоєнням із типом змінної, якій це значення надається, якщо виконується одна із таких умов:

- тип змінної, яка набуває значення виразу, і тип виразу однакові;
- тип змінної, яка набуває значення виразу, і тип виразу є сумісними порядковими типами, і діапазон значень типу виразу є піддіапазоном значень типу змінної;
- тип змінної, яка набуває значення виразу, і тип виразу є дійсними типами, і діапазон значень типу виразу є піддіапазоном значень типу змінної;
- тип виразу цілочисловий, а тип змінної, яка набуває значення виразу, дійсний;
- тип виразу і тип змінної, яка набуває значення виразу, є рядковим;

4- тип виразу є символьним, а тип змінної, котра набуває значення виразу, є рядковим.

Отже, у програмі дані одного типу можуть перетворюватися на дані іншого типу. Таке перетворення типів може бути явним або неявним. Явне перетворення типів здійснюється за допомогою вбудованих функцій, аргументи яких мають один тип, а значення, що вони їх повертають, мають інший тип (функції orcі, ігипс, гоипсі, сЬг, ріг). Також явне перетворення типів можна здійснити шляхом запису перед узятим у круглі дужки виразом, тип якого перетворюється, ідентифікатора типу, на який тип виразу потрібно перетворити, наприклад: ЬуЬеСТаїзе), іпіедег('X'), сііаг(60)). Якщо змінна *zz* має тип Ьуіе, кожне з присвоєнь сЬаг(22) : = 'A' та 22:=Ьуіе('A') надасть цій змінній значення 65, а присвоєння зНогТіпКгг) :=-100 -

значення 156 (максимальне значення типу Ьуіе дорівнює 255, вводиться значення -100, результат після перетворення типу становитиме 255 + 1 - 100 = 156).

У прикладі 2.10 демонструється принцип використання операцій явного перетворення типів. На рис. 2.9 наведено результати роботи програми ех2_5.

Приклад 2.10

```
ргодгат ех2_5;
 уаг
    і гґЬуаг: і піедег;
    зутЬоІуапсНаг;
Ьеділ
    УУГІІеіп('Сопуегііпд іурез');
    мгііе"1п(' Епіег іпіедег');
    геасЛп(іпіуаг):
    мгііе1п( 'Ьуіе(іпіуаг)=' .Ьуіе(іпіуаг)):
    мгііе1п('Епіег сіїагасіег');
    геасЛп(зутЬоІуаг);
    мгііеіп(' іпіедег(зутЬоїуаг)='.і піедег(зушЬо!уаг));
    мгііе1п(' сііаг(і піуаг+5)=' ,сИаг(іпіуаг+5));
    «гііе!п('Ьуіе(Та1зе)=',Ьуіе(Таїзе)):
    геасії п:
єпсі.
```

```
СопцеПіпд Іурез
Епіег Іпіедег
400
Ьуіе(іпіііаг)=144
Епіег сііагасіег
к
іпіедег (ЗуічЬо1иаг)=107
сИаг(іпИиаг+5)=-
ЬуІе(Еа1їе)=0
                                  і і
± и
```

Рис. 2.9. Результати роботи програми ех2_5. Перетворення типів

Неявне перетворення типів здійснюється при використанні в одному виразі або в одному операторі присвоєння сумісних, але різних типів даних.

Дійсні типи сумісні між собою за присвоєнням. Але якщо деяке дійсне значення виходить за межі діапазону допустимих значень певного дійсного типу, назвемо його К, то перетворення типу цього значення на К під час виконання програми призведе до помилки.

Натомість, якщо деяке ціле значення виходить за межі діапазону допустимих значень певного цілочислового типу, назвемо його *Т,* то перетворення типу цього значення на *Т* до помилки не призведе, але частина бітів у двійковому записі даного значення буде відкинута. Наприклад, значення виразу Ьуіе(260) дорівнює 4, оскільки 4 - це значення молодшого байта двохбайтового числа 260.

бясмовки

- **Інтегроване** середовище розробника Borland Pascal 7.0 містить текстовий редактор, транслятор, компонувальник, налагоджувач і довідкову систему.
- **Алфавіт мови** Pascal є підмножиною символів кодової таблиці ASCII.
- **Ідентифікатор** - це ім'я, яке вибирає користувач для позначення (ідентифікації) **елементів** програми (констант, змінних, типів, полів запису, процедур, функцій, модулів, програм). Він є послідовністю літер і цифр, що починається з літери.
- **Константа** — це елемент даних, значення якого не змінюється під час виконання програми.
- Програма, написана мовою Pascal, складається з двох частин: декларативної (оголошення ідентифікаторів, що використовуються в програмі) та операторної (опис виконуваних програмою дій).
- **Тип даних** визначає множину допустимих значень, які може набувати змінна або константа даного типу, множину допустимих операцій, що застосовуються до даних цього типу та способів зображення даних у пам'яті комп'ютера.
- **До простих** типів даних належать цілочислові типи, дійсні типи, символьний тип, булів тип, перелічуваний тип та інтервальний тип.
- **Усі прості типи,** крім дійсного, є порядковими. Множина допустимих значень будь-якого порядкового типу - це упорядкована та пронумерована послідовність. **Номери** значень таких типів обробляють функції zucc, pred і ord.
- **Для зображення** дійсних чисел в оперативній пам'яті використовується формат із плаваючою комою.
- **Змінна величина** — це узагальнення, абстракція якогось реального чи уявного об'єкта, що може перебувати в різних станах. Змінна позначається ідентифікатором. У програмуванні змінні використовуються для зберігання та передачі даних всередині програми.
- **Вираз визначає** послідовність перетворень значень, у результаті виконання якої утворюються нові значення.
- **Значення, до** якого застосовується операція, називається її операндом. Операції визначають дії, які потрібно виконати над операндами. Значення виразу обчислюється шляхом послідовного застосування операцій до операндів.
- **Оператори** визначають дії, що повинні здійснюватись комп'ютером під час виконання програми.
- **Надати значення** змінній можна за допомогою оператора присвоєння або процедур введення даних.
- **Оператор** присвоєння (:=) надає значення виразу, записаному праворуч від символу присвоєння, змінній, яка записана ліворуч від цього символу.
- **Введення даних** — це процес їх передачі із зовнішніх носіїв до комірок оперативної пам'яті для подальшої обробки. Виведення даних - це процес пе-

редачі їх з оперативної пам'яті на зовнішній носій. Введення даних здійснюють процедури геасі і геасїїп, а виведення — процедури мгіїе та мгіїеіп.

- Можливість використовувати операнди різних типів у виразах називається сумісністю типів; можливість присвоювати значення одного типу змінним іншого типу — сумісністю типів за присвоєнням.

Контрольні запитання та завдання

1. Яку структуру має ШЕ Вогіапсі РазсаІ 7.0?
2. Які файли потрібні для роботи з ШЕ Вогіапсі РазсаІ 7.0?
3. Які версії компілятора входять до складу ШЕ Вогіапсі РазсаІ 7.0?
4. Які різновиди лексичних одиниць є у мові РазсаІ?
5. Що визначає тип даних?
6. Які стандартні типи даних є у мові РазсаІ?
7. Що таке переповнення комірки оперативної пам'яті?
8. Охарактеризуйте порядкові типи.
9. Які вбудовані функції означені для типів Ьооїеап, іпіедег, геаі та сіїаг?
10. Чому типи Ьооїеап, іпіедег і сіїаг є порядковими, а геаі - ні?
11. Що являють собою типізовані константи?
12. Що таке вираз?
13. Які вбудовані функції можна використовувати в константних виразах?
14. Якими правилами визначається послідовність виконання операцій у виразах?
15. У чому полягає відмінність операцій CIIY та /?
16. Який тип має результат операції відношення?
17. Яку структуру має програма, написана мовою РазсаІ?
18. Які дії виконує оператор присвоєння?
19. У чому полягає відмінність між процедурами геасі та геасПп?
20. Які аргументи не можна використовувати у викликах процедур читання?
21. Що таке сумісність типів?

Вправи

1. Виразити операцію хог через інші булеві операції.
2. Знайти найбільше натуральне число, квадрат якого належить діапазону значень типу Іопді пі.
3. Обчислити значення виразів:
   а) (2*2=4) апсі ігие;
   б) (2*2=4) ог іаїзе;

в) (пої їгие) ог Таїзе;

г) (огсі(ТгиеМ) хог (огсКТа1зе)=0);

д) 2*огсі(Ігие)+3*огсІ(Та1зе);

е) Та1зе=1:гие=ТаІ5е;

є) 58 тосі 13 **сшу** 10;

ж) 9 тосі 5 * 12 **сшу** 16.

4. Якщо вказаний вираз коректний, обчислити його значення, а якщо ні, пояснити, чому:

а) 1=2=3;

б) (Тгие=Та1зе) апсі (їгие=Таїзе).

5. Вказати різницю між записами 0 і '0', А і 'А'.

6. Обчислити значення виразу:

а) сІіг(огсК'О') + 9);

б) сНг(огсК'А') + 25);

в) сЬгСогсК'О') - 16);

г) 'V > 'а';

д) огсК'9') - ОГСІСО[1]).

7. Обчислити значення логічних виразів для можливих значень змінних *х* та *у*.

а) (пої(х<5))апсі (пої(у>=7));

б) (поІ(х=у))ог(пої(у5));

в) по1((х<=8)апс1(у>4));

г) поТ((х>4)ог(у<=6)).

8. Замінити вирази з вправи 7 на еквівалентні, використовуючи закони Моргана, згідно з якими:

поЩумоваї) апсі (умова2)) еквівалентно (поі(умоваі)) ог (пої(умова2));

поЩумоваї) ог (умова2)) еквівалентно (поі(умоваі)) апсі (пої(умова2)).

9. Записати вирази, що перевіряють, чи є значення символьної змінної сії:

а) цифрою від '0' до '9';

б) маленькою латинською літерою;

в) латинською літерою (великою чи маленькою).

10. Записати вираз, що обчислює:

а) ціле число від 0 до 9, яке відповідає значенню символьної змінної сії (від '0' до '9');

б) символ від '0' до '9' за значенням цілочислової змінної сІдіТ від 0 до 9.

11. Припустимо, у змінній зушЬ символьного типу збережено маленьку латинську літеру. Записати вираз, що обчислює літеру, віддалену від кінця алфавіту настільки, наскільки літера у зутЬ віддалена від початку алфавіту. Наприклад, якщо зутЬ='а', то має бути виведений символ 'г', якщо зутЬ='Ь' — то 'у' і т. д.

12. У шістнадцятковій системі числення літерами A, B,..., P позначають десяткові числа 10, 11, ..., 15. Записати оператор, що обчислює:
    а) ціле число від 0 до 15 за значенням символьної змінної сґі, яким може бути цифра від '0' до '9' або літера від 'A' до 'P';
    б) символ від '0' до '9' або від 'A' до 'P' за цілим значенням змінної сід від 0 до 15.
13. Нехай а та Ь - імена цілочислових змінних. Записати арифметичний вираз, значенням якого є:
    а) значення наймолодшої цифри в десятковому зображенні а;
    б) сума значень цифр двозначного а (наприклад, для а = 83 ця сума дорівнює 11);
    в) сума значень цифр тризначного а (наприклад, для а = 123 ця сума дорівнює 6);
    г) більше з двох значень а і Ь.

ВКАЗІВКА

Функції обчислення максимального та мінімального з двох значень до складу стандартної бібліотеки функцій Pascal не входять; для розв'язання останнього завдання можна використати стандартну функцію, що обчислює модуль числа.

14. Нехай a, Ь, c,... - імена змінних цілих типів із додатними значеннями. Записати булів вираз, який є істинним лише за умови, що:
    а) a, Ь, c задають сторони трикутника;
    б) a, Ь, c задають сторони прямокутного трикутника;
    в) a, Ь, c задають сторони гострокутного трикутника;
    г) a, Ь, c задають сторони рівнобедреного трикутника;
    д) a, Ь, c задають сторони рівнобічного трикутника;
    е) a, Ь, c, сі задають сторони паралелограма;
    є) аі, Ьl, сі і а2, Ь2, c2 задають сторони рівних трикутників;
    ж) аі, Ьl, сі і а2, Ь2, c2 задають сторони подібних трикутників;
    з) цеглину розмірами *ах Ь х* с можна просунути до прямокутного вікна розміром *сі х е* так, щоб її грані проходили паралельно межам вікна.
15. Записати вираз, який є істинним тоді й тільки тоді, коли дві прямі, що задані цілими коефіцієнтами рівнянь вигляду $ax + by + c = 0$:
    а) паралельні та не збігаються;
    б) паралельні (можливо, збігаються);
    в) збігаються;
    г) перетинаються;
    д) перпендикулярні.

16. Підлога у прямокутній кімнаті розбита на *n x m* клітин. На дві клітини поставлені стовпи. Записати вираз, який обчислює ознаку того, що підлогу можна покрити дощечками розміром 2 x 1 клітини. Створити програму, у якій будуть вводитись розміри кімнати та координати клітин зі стовпами, а виводитись буде вказана ознака у вигляді булевої константи.
17. Є дві посудини *A та. B. B* посудині *A* міститься 1 л молока, а в посудині *B* — 1л чаю. Користувач вводить ємність склянки (у мілілітрах, не більше 1000 мл). З посудини *A* вичерпують склянку молока і переливають до посудини *B,* потім із посудини *B* вичерпують склянку суміші і переливають до *A* і т. д. — усього виконують 4 переливання. Програма має визначати, скільки в результаті в кожній із посудин міститиметься молока та чаю.
18. Користувач вводить деякий символ. Записати програму, що виводить зображену на рис. 2.10 фігуру, де замість символу «*» має бути символ, введений користувачем.

Рис. 2.10. Результат роботи програми до вправи 18

Розділ 3

Керування порядком обчислень

- Оператори розгалуження та вибору варіантів
- Поняття операторного блоку та вкладеності алгоритмічних конструкцій
- \+ Поняття циклу

  Три різновиди операторів циклу
- Процедури переривання циклів
- Приклади застосування операторів керування порядком обчислень
- Техніка організації циклів на основі рекурентних співвідношень
- Застосування ланцюгових дробів для обчислення деяких математичних функцій

3.1. Алгоритмічний вибір альтернатив

Алгоритмічна конструкція, що дозволяє виконавцеві алгоритму вибрати ту чи іншу послідовність дій залежно від певних умов, називається *розгалуженням* або *конструкцією вибору альтернатив.* У даному розділі розглянуто механізми використання цієї алгоритмічної конструкції у програмуванні. Є такі різновиди конструкції вибору, як вибір з двох альтернатив і поліваріантний вибір. Вибір із двох альтернатив розглядатиметься у розділах 3,1.1 та 3.1.2, а поліваріантний вибір - у розділі 3.1.4. Під час програмування деяких розгалужень виникає потреба у використанні операторних блоків, що розглядатимуться у розділі 3.1.3. У цьому ж розділі буде пояснено, як орієнтуватися в коді великих програм, що містять численну кількість конструкцій вибору та операторних блоків.

3.1.1. Вибір із двох альтернатив

Алгоритмічна конструкція альтернативного розгалуження, або конструкція вибору з двох альтернатив, дозволяє виконавцеві алгоритму вибрати один із двох варіантів дій залежно від істинності деякої умови. У мові РазсаІ альтернативні розгалуження реалізуються *умовним оператором (оператором розгалуження).* Синтаксис умовного оператора є таким:

ІТ <умова> -ИИеп <оператор!> [ЄІ5Є <оператор2>];

Тут іТ, ііієп, еізе — зарезервовані слова, що перекладаються як «якщо», «то», «інакше»; <умова> — довільний логічний вираз; <оператор1> і <оператор2> — довільні оператори.

Виконання умовного оператора починається з обчислення значення булевого виразу <умова>. Якщо цей вираз є істинним, то виконується <оператор1> і керування передається наступному за умовним оператору (<оператор2> пропускається). Якщо вираз <умова> є хибним, то <оператор1> пропускається, а виконується лише <оператор2> і на цьому дія умовного оператора вважається завершеною.

УВАГА

Символ «;» є символом, що відокремлює оператори. Оскільки умовний оператор завершується діями, які записані після слова еізе, то запис крапки з комою перед еізе є синтаксичною помилкою.

Зауважимо, що в синтаксисі умовного оператора фразу еізе <оператор2> записано у квадратних дужках, а отже, ця фраза є необов'язковою. Скорочена форма умовного оператора (без фрази еізе) реалізує одноальтернативне розгалуження. Якщо вираз <умова> в одноальтернативному розгалуженні є істинним, то <оператор1> виконується, якщо хибним — не виконується і на цьому дія умовного оператора вважається завершеною.

Нарешті, розглянемо детальніше умову, що записується після слова іі і являє собою певний логічний вираз. Нагадаємо, що логічний вираз може бути як окремим порівнянням або окремою булевою змінною, так і поєднанням порівнянь або булевих змінних за допомогою логічних операцій апсі, ог та поі. Наприклад, умова $0 < x < 5$ мовою РазсаІ записується так: (х >= 0) апсі (х <= 5), а умова $x < 0$ або $x > 5$ - так: (х <= 0) ог (х >= 5). Умови, записані за допомогою логічних операцій апсі та ог, називаються складеними, а умови, записані без таких операцій, - простими.

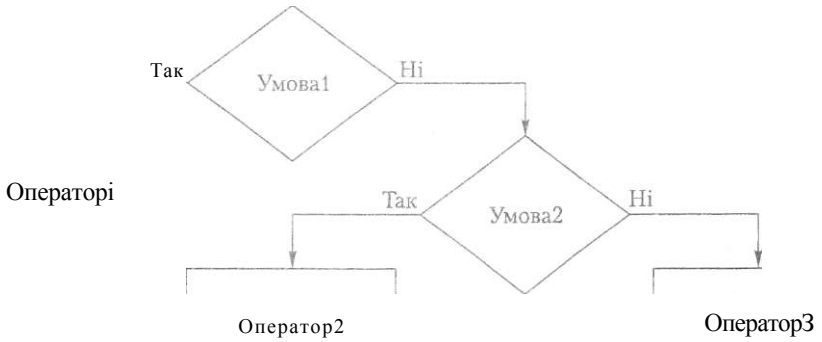
Наведемо найпростіший приклад застосування оператора вибору. Припустимо, потрібно визначити, чи належить значення дійсної змінної x проміжку [0;1], і вивести відповідне повідомлення. Ці дії виконує такий фрагмент програми:

```
іТ (х>=0) апсі (х<=1) Шеп
  мгіїеСх Ьеіопдз іо [0:1]')
еізе
  мгііеСх [illegible] пої Ьеіопд То [0:1]'):
```

3.1.2. Вкладеність конструкцій вибору

Гілки деякого розгалуження можуть містити інші розгалуження. У даному розділі для простоти розглянемо лише той випадок, коли одне розгалуження вкладене до гілки еі зе іншого. Зобразимо таку алгоритмічну конструкцію у вигляді блок-схеми (рис. 3.1) та наведемо синтаксис відповідного фрагменту Разсаі-програми.

```
іТ <умова1> їИеп <оператор1>
            еізе іТ <умова2> їііеп <оператор2>
                             еізе <оператор3>;
```



T

Рис. 3.1. Блок-схема розгалуження, вкладеного до іншого розгалуження

Зауважимо, що піраміди вкладених розгалужень завжди можуть бути реалізовані послідовними операторами розгалуження за рахунок ускладнення умов. Справді, легко побачити, що зображену блок-схемою з рис. 3.1 конструкцію виконують такі оператори:

```
if <умова1> Then <оператор1>;
if not(<умова1>) and <умова2> Then <оператор2>;
if not(<умова1>) and not(<умова2>) Then <оператор3>;
```

Але слід зазначити, що вкладені умовні оператори працюють значно швидше, ніж серія умовних операторів у скороченій формі, завдяки тому, що робота всієї конструкції завершується автоматично після виконання однієї з умов. Додаткове прискорення може бути досягнуте за рахунок запису умов, що перевіряються частіше, нагорі піраміди вкладених розгалужень. Приклад 3.1 демонструє застосування вкладених розгалужень. На рис. 3.2 зображено блок-схему алгоритму розв'язання розглянутої в прикладі задачі, а на рис. 3.3 - результати роботи програми, що реалізує цей алгоритм.

Приклад 3.1

Потрібно обчислити значення функції *y(b, c, x):*

$$y = \begin{cases} bx + c, & x < -4; \\ bx/c, & x > 4 \text{ і } c \neq 0; \\ bx^2, & -4 < x < 4; \\ 1, & \text{в інших випадках.} \end{cases}$$

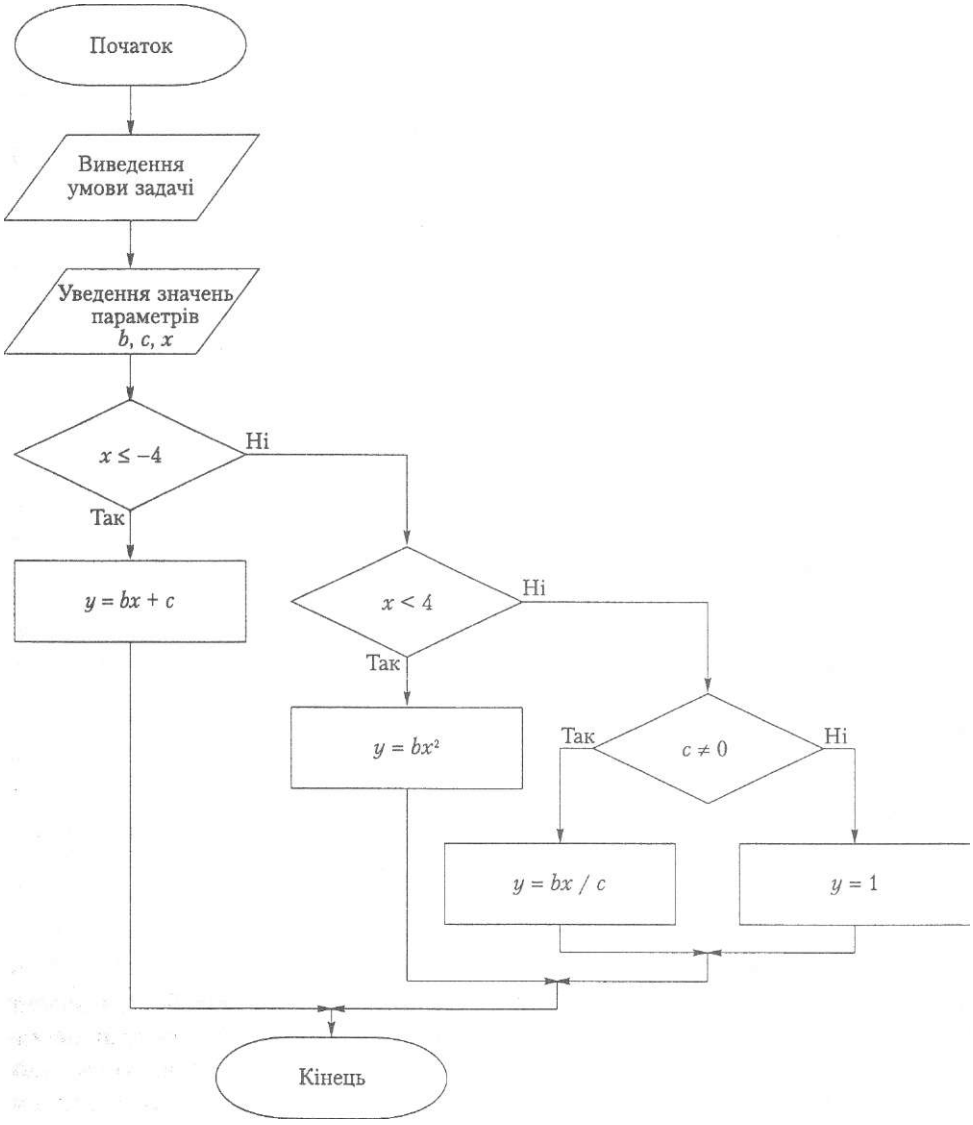


Рис. 3.2. Блок-схема алгоритму обчислення значення функції за різними умовами

Складемо програму за наведеною на рис. 3.2 блок-схемою. Значення x^2 в мові РазсаІ обчислює стандартна функція здг(х). Детальніше про стандартні функції йтиметься в розділі 4.3.1.

```
ргодгат ех3_1:
маг
 y.b.c,x:rea1;          {змінні для результату та вхідних даних}
Ьедіп
```

```
мгіТе1п('іприї у:');                          {умова задачі}
игіТеІпС'у:=Ь*х+с.     іТ х<=- 4');
мгіїе1п('у:=Ь*5Яг(х),  IT (х<4)апсі(х>-4)');
мгіТе!п('у:=Ь*х/с,     іТ (х>-4)апсі(с<>0)[1]);
мгіТе1п( 'у :=1,        іТ(с=0)апсі(х>4)');
мгіГе('ілриТ уагіаЬІез Ь.с.х:[1]);
геасПп(Ь.с.х):              {увести дані для розрахунків}
IT х<=-4 їіпеп у:=Ь*х+с          {обчислити перший вираз}
         еізе іТ (х<4) апсі (х>-4)
                IЬеп
                  у:=Ь*здг(х)    {обчислити третій вираз}
                еізе іТ с<>0
                       їііеп
                         у:=Ь*х/с  {обчислити другий вираз}
                       еізе у:=1;
мгііеІпС Ь=',Ь:6:2,' с=',с:6:2,' х=',х:6:2):
мгіТе1п(' гегиіі; : у=',у:6:2);          {вивести результат}
єпсі.                                      {кінець програми}
```

```
йеГіпе у:
•у:=Ь*х+с,     ІГ х<=- 4
1):-Ь«5цс(х),  ir (х<Ц)апгі(х>-ї+)
у:»Ь»х/с,      ir (х>-М)апс!(с<>0)
     ,         іГ(с = 0)ані1(х>іі)
іприї иагізЬ1е5 Ь,с,х:3 2 1
 Ь-  3.00 с=  2.00 х-  1.00
гегиН :      3.00
```

а

РИС. 3.3. Результати роботи програми ех3_1. Вкладені розгалуження

3.1.3. Операторний блок

Синтаксис умовного оператора, що його було розглянуто в розділі **3.1.1, дозволяє** виконувати лише один оператор у разі істинності певної умови і лише **один оператор** у разі її хибності. Проте часто виникає потреба у розгалуженнях, **гілки яких** містять більше ніж одну інструкцію. В такому разі застосовують **операторні блоки.**

Операторний блок, або *складений оператор,* — це послідовність **операторів, що** розпочинається ключовим словом Ьеді п та завершується **ключовим словом єпсі** (слова Ьеді п та єпсі інколи називають оиераторними дужками). **Операторний блок** може перебувати в будь-якому місці програми, де синтаксисом **мови припускається** наявність оператора. Синтаксис операторного блоку має такий **вигляд:**

```
Ьеді п
    <оператор1>;

    <оператор2>;
єпсі;
```

Усередині оиераторного блоку можуть міститися довільні оператори, у тому числі й складені, вони виконуються у порядку запису. Використання операторних блоків у гілках іНеп та e1$e оператора розгалуження продемонструємо на прикладі програми, що обчислює корені квадратного рівняння.

Приклад 3.2 _ _ ___

Необхідно знайти розв'язки квадратного рівняння $ax^2 + Ьx + c = 0$, коефіцієнти якого є дійсними числами, що їх вводить користувач. Залежно від значень коефіцієнтів *а, Ь, с* та дискримінанта *ci* = $Б^2$ - 4*ac* можливі такі результати: всі дійсні числа є коренями *(а* - 0, *Ь* = 0, с = 0), коренів немає (а = 0, *Ь* = 0, *с Ф* 0), є один корінь (а = 0,Ьф 0), є два різних дійсних корені (аФО, й> 0), два дійсних корені збігаються (аФ0,с1=0) або існує два комплексно-спряжених корені (аф0,сі< 0).

Отже, для коректного розв'язання даної задачі слід виконати декілька порівнянь із нулем коефіцієнтів рівняння та його дискримінанта. Ці порівняння можна виконувати у різній послідовності, і відповідно до цього можна складати різні алгоритми. Алгоритми розв'язання однієї задачі можуть відрізнятися за своєю ефективністю. Припустимо, що для даної задачі найефективнішим є той алгоритм, який мінімізує середню кількість порівнянь для довільного набору вхідних даних. Спробуємо побудувати такий алгоритм. Зауваживши, що перевірка умови *а* = 0? відбувається в усіх варіантах розв'язання, робимо висновок, що розгалуження за цією умовою слід виконувати найпершим, а перевірку всіх інших умов здійснювати на його гілках. У тому разі, коли *а* = 0, перевірка умови *с* = 0? є доцільною, лише коли *Ь* - 0, тому умову *Ь* = 0? слід перевіряти раніше, ніж умову *с* = 0?. Врахувавши ці міркування, запишемо алгоритм і програму розв'язання квадратного рівняння.

Алгоритм обчислення коренів квадратного рівняння

1. Увести коефіцієнти квадратного рівняння *а, Ь, с.*
2. Якщо *а* <> 0, то виконати дії 2.1-2.4.
   2.1. Обчислити дискримінант за формулою *i=Ь*^{2} - *А ас.*
   2.2. Якщо *ci* > 0, то обчислити корені *X* та x_2 за наведеною нижче формулою та вивести їх.

$$x_{1,2} \quad \frac{-Ь \pm 4д.}{2\,a}$$

   2.3. Якщо *ci* < 0, то вивести повідомлення «Корені комплексні».
   2.4. Якщо *ci* - 0, то вивести повідомлення про рівність двох коренів та обчислити їх значення за формулою $x_{i\,2}$ = *-Ь/2а.*
3. Якщо *а* = 0, то виконати дії 3.1-3.3.
   3.1. Якщо *Ь Ф* 0, то обчислити й вивести значення *-с/Ь.*
   3.2. Якщо *Ь* = 0 і с *Ф* 0, то вивести повідомлення «Коренів немає».
   3.3. Якщо *Ь* = 0 і с = 0, то вивести повідомлення «Усі дійсні числа є коренями».

Наведемо програмне розв'язання задачі обчислення коренів квадратного рівняння,

```
ргодгат ех3_2;
уаг a.b.c.сіїгеаІ;    {коефіцієнти квадратного рівняння та дискримінант}
    x!,x2:rea1;                          {корені квадратного рівняння}
Ьеділ
  мгіїеІпС 'зоІиТіоп оТ диасігаїіс едиаТіоп');
  нгіїеІпС'епїег коеТТісіепІз:  a.b.c');
  геасПп(a.b.c);                        {ввести значення коефіцієнтів}
  іТ  a<>0
    іілеп
      Ьеділ
        сІ:=5дг(b)-4*a*c:                      {обчислити дискримінант}
        ІГ  d>0
          їііеп
            Ьеділ
              x1:=(-b+здгШ))/(2*a);
              x2:=(-b-здгШ))/(2*a);
              ипїе1п([1]x1=[1],x1:6:2.'  x2=' , x2:6:2);
            єпсі
          еізе
            іТ  d=0
              їИеп
                мпїе1п( 'гооїз аге ериаі:  x-',-b/(2*a):6:2)
              еізе
                мгіІе1п('сотріех гооїз')            {дискримінант < 0}
      єпсі                                        {кінець гілки a<>0}
    еізе                                                  {a=0}
      іТ  b<>0
        ІИеп мпТе1п('x=[1] ,-c/b:6:2)                {лінійне рівняння}
        еізе
          іТ c<>0                {рівняння типу "ненульове число = 0"}
ТНеп мгіГе1п('по гооїз')
            еізе мгіІе1п('а11  геаі  пишЬегз аге  Ібе  гооїз');
                                                  {рівняння типу  0 = 0 }
єпсі.
```

```
їоіиїіоп оГ  ^иа()^-а^:іс  ечиаїіоп
епіег  коеГГісіепїз:  а,Ь,с
12  1
гоо!5  аге  ециаі:  х»  -1.00
```

Л

РИС. 3.4. Результати роботи програми ех3_2. Розв'язання квадратного рівняння

У коді програми ех3_2 використано два операторних блоки, призначених для структуризації декількох умовних операторів. У більш складних програмах кількість і розмір вкладених один до одного умовних операторів та операторних блоків може стати на порядки вищою. У зв'язку з цим постають питання: як для

операторної дужки єпсі визначати відповідну їй дужку Ьеділ та як за фразою еізе визначати відповідну їй фразу і Т? Відповідь на ці два питання сформулюємо у вигляді такого правила.

Присвоїмо лічильнику нульове значення. Починаючи від слова епс! (еі зе), проглядатимемо текст програми у зворотному порядку. Якщо зустрічаємо інше слово єпсі (еізе), то збільшуємо лічильник на одиницю, якщо зустрічаємо слово Ьеділ (іТ) — то на одиницю зменшуємо. Слово Ьеділ (ІТ), на яке натрапимо при нульовому значенні лічильника, і буде відповідати тому слову єпсі (еі зе), від якого було розпочато пошук. Якщо під час пошуку слова і Т зустрілось слово єпсі, то весь операторний блок, який цим словом завершується, слід пропустити, не враховуючи розташовані всередині нього умовні оператори. Зауважимо, що словом єпсі може завершуватися не лише операторний блок, але і оператор вибору сазе, який буде розглянуто у наступному розділі. Тому під час процедури пошуку слово сазе вважається ідентичним слову Ьеділ.

Це правило дозволяє легко інтерпретувати конструкцію, в якій один умовний оператор вкладено до фрази іііеп іншого. Розглянемо приклад такої конструкції.

```
іТ <умова1> їИеп іТ <умова2> їНеп <оператор2>
                              еізе <оператор3>;
```

Хибність якої з умов, <умова1> чи <умова2>, приведе до виконання оператора <оператор3>? Іншими словами, фраза і Т <умова2> іііеп <оператор2> є скороченою формою умовного оператора чи частиною оператора іТ <умова2> ІИеп <оператор2> еізе <оператор3>? Застосувавши наведене вище правило, визначимо, що оператор <оператор3> буде виконано в разі хибності умови <умова2>. Розглянемо інший приклад.

```
іТ <умова1> Тіїеп Ьеділ іТ <умова2> їііеп <оператор2> єпсі
            ei5e <оператор3>;
```

Керування перейде до оператора <оператор3> у разі невиконання умови <умова1>, оскільки блок Ьеділ ІТ <умова2> іНеп <оператор2> єпсі інтерпретується як один складений оператор.

3.1.4. Поліваріантний вибір

Узагальненням альтернативного розгалуження є алгоритмічна конструкція поліваріантного вибору, що дозволяє виконувати одну з декількох алгоритмічних гілок залежно від значення деякого виразу. У мові РазсаІ цю алгоритмічну конструкцію реалізовано *операторомвибору.* Наведемо його синтаксис:

```
сазе <селектор> оТ
   <список констант1>:<оператор1>;
   <список констант2>:<оператор2>;

   [еізе <операторп>:]
єпсі:
```

Тут сазе, оТ, еізе — це зарезервовані слова, що перекладаються як «випадок», «з», «інакше»; <селектор> — змінна або вираз, який має довільний перелічуваний тип;

<список констант> - перелік розділених комами значень того самого типу, що і селектор; <оператор> — будь-який оператор; єпсі - кінець оператора сазе.

Оператор вибору виконується за таким алгоритмом. Спочатку обчислюється значення виразу-селектора. Потім вибирається той список констант, до якого належить отримане значення, виконується відповідний оператор і на цьому дія оператора сазе завершується. Якщо поточне значення селектора не збігається з жодною з констант вибору, то виконується гілка еізе, а якщо її немає, то виконання оператора вибору завершується.

УВАГА

Тип селектора має бути перелічуваним. Неприпустимим є використання селекторів дійсних або структурованих типів.

Приклад 3.3

Запрограмуємо калькулятор, що виконує чотири арифметичні дії над дійсними числами. Користувач вводить із клавіатури символ операції та значення операндів, а програма повинна обчислити результат арифметичної дії. Для зберігання значень операндів використаємо дійсні змінні орегапсії та орегапс!2, символ операції зберігатимемо у змінній орегаііоп символьного типу, а результат — у змінній -Є51ІІІ дійсного типу. Про введення некоректного символу операції та про спроби ділення на нуль сигналізуватиме логічна змінна Лад. Результати роботи програми калькулятора наведено на рис. 3.5.

```
Рrодгат ех3_3:
уаг орегаїїоп:сИаг;                    {символ арифметичної операції}
    гезиїї.орегапсії,орегапсі2:reai;     {результат операції, операнди}
    Т1ад:Ьоо1еап;              {ознака некоректного символу операції}
Ьеділ
                                                    {введення даних)
 мп1е1п('саісиїаіог ');
 ягіІеС'епіег орегапсії :');          геасЛп(орегапсіІ);
 игіїеС'епіег орегаііоп + - * / :'); геасПп(орегаііоп);
 мгіїеС 'епіег орегапс!2 :');          геасії п(орегапсі2);
 Лад:=1;гие;
 сазе орегаїіоп оТ            {аналіз символу арифметичної операції}
                                    {виконання арифметичних дій}
   '*':   гези1і:=орегапсІ1*орегапс12;
   '+':   гези11:=орегапсі1+орегапсі2;
   '-':   гєзи1ї:=орегапсі1-орегапсі2;
   '/':   іі орегапс!2<>0                  {запобігання діленню на нуль}
              Їліеп гези1і::=орегапсІ1/орегапсі2
              еізе
                   Ьеділ
                     мгіІеІпСсіІУізіоп Ьу 2ЄГ0');
                     Лад:=Та1зе:                  {ознака помилки}
                   епсі;
```

```
EI 5€
    Ьеділ
     T1ад:=TaI5e:                    {уведено помилковий символ операції}
     игіIe1п('іт/a1icI   operaiion[1]);
    єпсі;
єпсі;
                                     {виведення результату}
іТ TIадоТаїзе IHеп мгііе1п([1] гези1і=',гезиі 1::б:3)
              еізе «гі"Ьеіп('гезиН поі сIе"Гіпєсі');
єпсі.
```

```
caicиїaior          ^   '   "   ».|
enier operanйi :2                 ||
enier operaiion + - * / : /
enier operanйr
гезиіі» B.50O
```

ї й *±Ш*

РИС. 3.5. Результати роботи програми ех3_3. Калькулятор

3.2. Алгоритмічна конструкція повторення

У розглянутих вище прикладах усі оператори виконувалися не більше одного разу. Проте часто постає потреба виконати один і той самий оператор декілька разів. Для цього застосовують *оператори циклів,* що реалізують алгоритмічну конструкцію повторення. Цикл складається із заголовка та тіла. У *заголовку циклу* Зазначається *умова завершення циклу,* а *тіло циклу* являє собою блок операторів, що повторюються. Кожне виконання операторів тіла циклу супроводжується перевіркою умови завершення циклу і називається його *ітерацією.* Якщо умова завершення хибна, то тіло циклу виконується ще раз, якщо істинна, то виконання циклу припиняється і здійснюється перехід до виконання наступного за циклом оператора. Замість умови завершення циклу може перевірятися її заперечення — *умова продовження,* хибність якої призводить до припинення виконання циклу, а істинність — до продовження. Змінні, значення яких модифікуються в тілі циклу і впливають на істинність умови завершення, називаються *параметрами циклу.* Виконанню будь-якого циклу має передувати ініціалізація його параметрів.

У мові РазсаI є три різновиди операторів циклу: оператор циклу з передумовою, оператор циклу з постумовою та оператор циклу з лічильником. Вибір того чи іншого оператора циклу для розв'язання певної алгоритмічної задачі здійснюється залежно від типу умови завершення циклу. Цикли з передумовою розглянуто в розділі 3.2.1, цикли із постумовою - в розділі 3.2.2, а цикли з лічильником — у розділі 3.2.3. Нарешті, у розділі 3.2.4 обговорюються допоміжні засоби керування циклами — процедури Ьгеак і **соniiпие.**

3.2.1. Цикл із передумовою

Насамперед визначимо відмінні риси, що вирізняють цикл із передумовою з-поміж інших циклічних операторів. У циклі з передумовою перша перевірка умови продовження циклу відбувається ще до першого виконання його тіла. Це означає, що за деяких значень параметрів циклу його тіло може не виконатися жодного разу. Саме за цією ознакою під час розв'язання певних задач віддають перевагу циклу з передумовою над циклом із постумовою. З іншого боку, і цикл з передумовою, і цикл із постумовою застосовують лише у тому випадку, коли кількість повторень є невідомою до початку виконання циклу — в іншому разі використовують цикл із лічильником. Отже, наведемо синтаксис оператора циклу з передумовою:

```
мИіІе <умова продовження циклу> йо
   <оператор>;
```

ТутмИіІе <умова продовження циклу> сіо є заголовком циклу, <оператор> — його тілом. Тіло циклу може бути операторним блоком і містити в собі будь-які оператори: циклу, вибору, присвоєння тощо. Умова продовження циклу повинна бути виразом булевого типу. Оператору циклу з передумовою відповідає блок-схема, що зображена на рис. 3.6.

Рис. 3.6. Блок-схема циклу з передумовою

Оператор циклу з передумовою виконується за таким алгоритмом. Спочатку обчислюється умова продовження циклу, що записана в його заголовку. Якщо вона істинна, то виконується тіло циклу, інакше виконання циклу припиняється. Після виконання тіла циклу буде знову перевірена умова його продовження. Чергування виконання тіла циклу та перевірки умови продовження триває доти, доки умова не стане хибною.

УВАГА

Згідно з синтаксисом оператора \n/liile тіло циклу е одним оператором. Для того щоб в циклі виконувалося декілька операторів, їх треба оточити операторними дужками Ьеділ...єпсі.

Приклад 3.4__

Розглянемо задачу перевірки натурального числа на простоту. Як відомо, число *n* є простим, якщо воно ділиться лише на одиницю та на себе, тобто його додатними дільниками є лише числа 1 і *n,* інакше *n* вважається складеним числом. Отже, в алгоритмі перевірки числа на простоту повторюються такі дії: перевірка подільності числа *n* на дільник *к* та вибір наступного дільника шляхом збільшення *к* на одиницю. Таким чином, алгоритм є циклічним. Очевидною умовою продовження циклу є умова *к* < *n.* Оскільки на одиницю ділиться будь-яке число, то найменше значення параметра циклу *к* дорівнюватиме 2. Неважко побачити, що перебір всіх *n* - 2 чисел, від 2 до *n* - 1, є недоцільним, оскільки значення найбільшого дільника складеного *n* не може перевищувати величини *-in.* У такому разі достатньо перебрати лише цілі числа від 2 до ^ (символами позначатимемо цілу частину числа *x).*

Розглянемо будову циклу детальніше. Як сигналізувати про те, що знайшовся деякий дільник числа *n?* Це можна зробити за допомогою змінної логічного типу, дамо їй ім'я TI ад (прапорець). До початку виконання циклу присвоїмо цій змінній значення Тгие, а в разі знаходження дільника *n* - значення Таїзе. Отже, після виконання циклу ТІ ад дорівнюватиме Таїзе, якщо дільник *n* знайдено (*n* - складене), і значенням Тіад залишиться Тгие, якщо *n* є простим. Зауважимо, що в разі знаходження хоча б одного дільника *n* подальша перевірка чисел не потрібна, оскільки *n* тоді напевне буде складеним. Таким чином, цикл можна зупиняти не тільки по досягненні дільником *к* значення I •*4n* I, а й внаслідок хибності значення змінної Тіад. Умова продовження циклу буде складеною: (к<=Тгипс(5дгI(п))) апсі Тіад (вираз Тгипс(здгі(п)) дорівнює цілій частині кореня з *n;* функції Тгипс(х), здгі(х) та інші стандартні функції розглядатимуться в розділі 4.1.3). Залишилося навести код програми та екранну копію результатів її виконання (рис. 3.7).

```
ргодгат ех3_4;                          {визначити прості числа}
уаг п.к:іпТедег;                      {число, потенційний дільник}
    Т1ад:Ьоо1еап:                      {ознака наявності дільника}
Ьеділ
   мгіТе1п('пишЬег зішріісіїу');
   мгіТе1п('епіег іпіедег: '):
   геасііп(п):                                      {увести число}
   к:=2:                                  {вибрати перший дільник}
   Лад:=ігие;          {ще немає підстав вважати п складеним}
   {перебирати потенційні дільники}
   мНіїе (к<=1гипс($ягї(п))) апсі Лад сіо
   Ьеділ
     іТ п тосі к = 0 ТЬеп                    {якщо к ділить п.}
       Лад:=Та1зе;                        {сигналізуємо про це}
     к:=к+1;                    {наступний потенційний дільник}
   єпсі:
   іТ Лад їИеп мгіТе1п(п,' із зішріе')          {число просте}
          еізе мгіТеїпСп,' із по! зішріе'); {число складене}
єпсі.
```

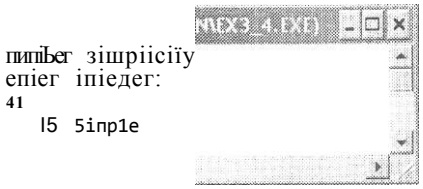


РИС. 3.7. Результати роботи програми ex3_4. Перевірка числа на простоту

Оскільки цикл може почати роботу лише в разі істинності умови його продовження, а завершити роботу - лише в разі хибності цієї умови, то її істинність, а отже, і значення параметрів циклу, повинні змінюватися під час його роботи. В іншому разі відбудеться «зациклення», тобто виникне ситуація, коли цикл ніколи не завершує своєї роботи. У середовищі Borland C++ 7.0 виконання нескінченного циклу, а разом із ним і виконання всієї програми переривається комбінацією клавіш Ctrl+Break.

3.2.2. ЦИКЛ ІЗ постумовою

Як і цикл із передумовою, цикл із постумовою застосовують тоді, коли кількість ітерацій циклу є невідомою до початку його виконання. Умова завершення циклу з постумовою записується після тіла циклу та вперше перевіряється після виконання операторів тіла. А отже, цикл з постумовою за будь-яких обставин буде виконано принаймні один раз - в цьому і полягає його головна відмінність від циклу з передумовою. Синтаксис оператора циклу з постумовою такий:

```
repeat
   <оператор1>; ... <оператор(>i>;
until <умова завершення циклу>;
```

Тут repeat, until - зарезервовані слова, <оператор1>;...<оператор№>; - тіло циклу; <умова завершення циклу> - деякий булів вираз. Дослівно ця мовна конструкція перекладається так:

```
повторювати
   послідовність операторів
доти, доки не виконається умова
```

Блок-схема циклу з постумовою зображена на рис. 3.8.

Оператор циклу з постумовою працює за таким алгоритмом. Спочатку виконуються оператори, що входять до складу тіла циклу. Потім обчислюється умова завершення циклу. Якщо вона істинна, цикл завершує свою роботу, інакше повторюється виконання його тіла. Процеси виконання тіла циклу та перевірки умови завершення чергуються доти, доки умова не стане істинною. Зауважимо, що параметри циклу з постумовою, як і циклу з передумовою, повинні змінюватись під час його виконання так, щоб не трапилось «зациклення».

Рис. 3.8. Блок-схема циклу з постумовою

Синтаксичною особливістю оператора циклу з постумовою є те, що тіло циклу не обов'язково оточувати операторними дужками Ьеді п та єпсі, оскільки воно розташоване між зарезервованими словами гєреаг та ипіі 1. Символ «;» після останнього оператора тіла циклу також є необов'язковим, що уможливлює запис циклу із порожнім тілом. Наведемо приклад застосування даної циклічної конструкції.

Приклад 3.5

Запрограмуємо гру «вгадування числа». Програма генерує випадкове ціле число з деякого діапазону, а користувач намагається його відгадати, вводячи значення з клавіатури. Якщо число вгадане, то програма виводить повідомлення про це і завершує свою роботу, інакше спроби відгадати число повторюються. На початку виконання програми встановлюється певний «призовий фонд», що зменшується з кожною невдалою спробою користувача.

Очевидно, що процес відгадування є циклічним, оскільки повторюються такі дії, як введення числа, його порівняння із згенерованим і зменшення призового фонду. Для моделювання цієї гри найзручнішою конструкцією буде саме цикл із постумовою, оскільки, по-перше, кількість спроб наперед невідома, і, по-друге, принаймні одна спроба повинна бути здійснена. Використаємо змінні для збереження «призового фонду» (ргі ге), числа, що буде згенероване комп'ютером (у), числа, що його вводитиме користувач (х), та штрафу (репаїіу).

Для генерування випадкового числа використовуються дві бібліотечні підпрограми. Процедура гапсіотіге ініціалізує датчик випадкових чисел значенням, отриманим від системного таймера, а функція гапсіот(п) повертає згенероване цим датчиком випадкове ціле число з діапазону 0 .. *п*- 1. Процедуру гапсіотіге слід викликати до першого виклику функції гапсіот. Насправді згенеровані в такий спосіб числа не є випадковими з математичної точки зору, вони лише схожі на випадкові і називаються *псевдовипадковими числами.* Результати роботи програми вгадування псевдовипадкового числа наведено на рис. 3.9.

```
ргодгат ех3_5;
сопзі репа11у=10;                                    {штраф}
уаг х,                        {число, що вводиться з клавіатури}
    у,                          {згенероване комп'ютером число}
    ргі2е:іпіедег;                 {кількість "призових" балів}
Ьеді п
  мгіІе1п('сieTine пигльег, мЬісН сотриіег депегаїе');
  гапсіотіге;        {і ні ці ал і зувати генератор випадкових чисел}
  у:=гапсіот(1000);    {генерувати випадкове число з діапазону
                                                  від 0 до 999}
  ргі ге:=100;                        {початкова "призова сума"}
  гереаі
    мгіІе1п('епіег пишЬег');
    геасПп(х);                       {увести число з клавіатури}
    іТ х>у "ЬИеп Ьеді п
                мгіі:е1п('Мгопд. епіег зтаїїег');
                рн2е;=рп2е-репа11у;
              єпсі;
    іТ х<у ІНеп Ьеді п
                мгіІе1п('Игопд, епіег Ьіддег');
                ргі ге;=ргі ге-репаїІу;
              єпсі;
  иііі х=у;              {ПОКИ ЧИСЛО не відгадане користувачем}
  мгіІеІпСУои ііауе диеззесі гідіїї!');
  мгіїеІпСУои ііауе моп ІНе ргі ге ',ргіге);
єпсі.
```

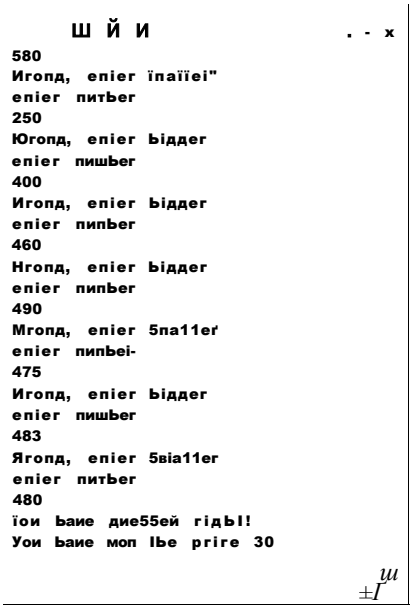


РИС. 3.9. Результати роботи програми ех3_5. Вгадування псевдовипадкового числа

3.2.3. Цикл із лічильником

Розглянемо задачу обчислення факторіала невід'ємного цілого числа *n*. Нагадаємо формулу факторіала: *n*! = 1 • 2 •... • *(n* -1) • *n,* де *n* > 0, 0! = 1. Легко побачити, що *n*! = (я - 1)! • *n,* тому обчислення можна звести до багаторазового виконання операції Тасіогіаі :=Тасіогіа1*і, де і = 2, 3,..., *n,* а початкове значення змінної TacTorIaI дорівнює 1. Отже, задачу можна розв'язати за допомогою циклічної структури. Тіло циклу складатиметься із однієї вищезгаданої операції присвоєння, а кількість її повторень становитиме *n.* Таким чином, умовою завершення циклу буде виконання певної кількості ітерацій. Відстежити істинність такої умови дозволяє спеціальний різновид параметра циклу, *лічильник ітерацій.* Лічильник — це змінна, що під час кожного повторення збільшується на одиницю. В *операторі циклу з лічильником* облік числа виконаних ітерацій ведеться автоматично, і тому цей оператор є зручною формою запису циклів із наперед визначеною кількістю повторень. Наведемо синтаксис оператора циклу з лічильником, а нижче, на рис. 3.10, - його блок-схему.

```
Тог <лічильник>:=<початкове значення> То <кінцеве значення> сіо
    <оператор>:
```

Тут Тог, То, сіо — зарезервовані слова («для», «до», «виконати»); <початкове значення> та <кі нцеве значення> — вирази того самого перелічуваного типу; <лі чильник> — змінна перелічуваного типу, що є узгодженим за присвоюванням з типом початкового та кінцевого значення; <оператор> — простий або складений оператор, що є тілом циклу. Фраза від слова Тог до слова сіо є заголовком циклу, а зазначений після слова сіо оператор — тілом циклу.

Виконання оператора Тог здійснюється за таким алгоритмом. Спочатку обчислюються та порівнюються значення виразів початкове значення> та <кінцеве значенням Якщо початкове значення більше за кінцеве, то виконання циклу завершується, інакше лічильнику циклу присвоюється початкове значення і виконується тіло циклу. Після виконання тіла циклу порівнюється поточне значення лічильника з його кінцевим значенням, і, якщо ці значення не рівні, то поточне значення лічильника збільшується на одиницю і виконання циклу триває далі. Цикл завершує свою роботу тоді, коли поточне значення лічильника циклу стає рівним його кінцевому значенню.

УВАГА

Збільшення лічильника здійснюється автоматично, тобто на кожній ітерації циклу неявно виконується оператор <лічильник>:=<лічильник>+1, а отже, цей оператор не потрібно записувати до тіла циклу в явному вигляді.

Цикл з лічильником не завершить своєї роботи, якщо поточне значення лічильника буде більшим за кінцеве значення. Наприклад, цикл Тог і:=1 Іо 5 сіо і:=6; не зупиниться ніколи. Отже, змінювати значення лічильника всередині тіла циклу вкрай небезпечно.

У мові РазсаІ є ще й другий різновид оператора циклу з лічильником. Цей оператор на кожній ітерації циклу автоматично зменшує значення його лічиль-

ника на одиницю. Синтаксис такого оператора відрізняється від синтаксису оператора циклу, який збільшує лічильник, лише тим, що слово іо замінено словом скмгЬо:

```
іог <лічильник>:=<початкове значення> сіошіо <кінцеве значення> сіо
    <оператор>;
```

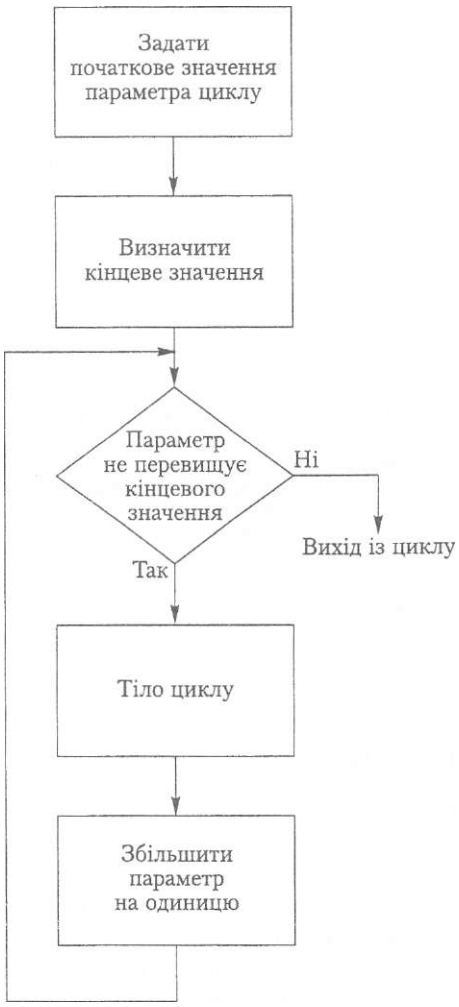


Рис. 3.10. Блок-схема циклу з лічильником

Якщо початкове значення лічильника циклу буде меншим за кінцеве, то цикл Тог.-сІсмЛо не виконається жодного разу. Якщо ці значення рівні, то цикл виконається лише один раз.

Приклад 3.6_

Складемо програму обчислення факторіала цілого невід'ємного числа. Тіло циклу в цій програмі міститиме один оператор — присвоєння змінній Tacioriai добутку її попереднього значення та поточного значення лічильника циклу. Результати роботи програми наведено на рис. 3.11.

```
Ргодгат ех3_6;                    {обчислення факторіала числа}
 уаг Tacioriai:IопдіпТ;                  {значення факторіала}
     п,і:іпТедег;            {вхідне число та лічильник циклу}
 Ьеділ
   мгіТе1п('TacToriaI саісиїаїіоп - п! '):
   гереаі                                        {введення даних}
     мгіТе('епіег іпіедег пишЬег:');
     геасЛп(п);
   ипііі п<=16;             {обмеження діапазону вхідних даних}
   TacToriaI:=1;                {початкове значення факторіала}
   Тог і:=2 То п сіо    {заголовок циклу обчислення факторіала}
      Tacioriai :=TacTопа1*і;                       {тіло циклу}
   мгіТе1п(п,'! = '.Tacioriai);           {виведення результату}
 єпсі.
```

Допустимий діапазон вхідних значень було обмежено числом 16, оскільки функція факторіала числа дуже швидко зростає, і вже для значення n =17 величина n\ перевищує найбільше допустиме значення типу IопдіпТ.

```
Гacioriai caicиїaiion - п!
enier inieder пиinber:9
I9»-362880
```

РИС. 3.11. Результати роботи програми ех3_6. Обчислення факторіала числа

3.2,4. Переривання циклу

У деяких програмах виникає потреба завершити цикл або його ітерацію передчасно. Де завжди можна зробити, ускладнивши умову завершення циклу або записавши додаткові оператори розгалуження до його тіла. Проте такі дії можуть зробити код програми надто громіздким. У мові РазсаІ цього ефекту можна досягти простіше, використовуючи процедури Ьгеак та сопТіпие.

Передчасний вихід із циклу означає, що умова завершення циклу під час виходу з нього є хибною. Для примусового переривання циклу слід викликати процедуру Ьгеак. Процедура сопТіпие здійснює пропуск усіх інструкцій, записаних за її викликом в тілі циклу. Таким чином, після виклику процедури сопТіпие завжди перевіряється умова завершення або продовження циклу.

Якщо використовуються вкладені цикли, то процедура Ьгеак перериває той цикл, у якому вона викликається. Після переривання внутрішнього циклу управ-

ління передається заголовку зовнішнього циклу для перевірки умови його продовження. Аналогічно працює і процедура соп Тіпие: її виклик забезпечує виконання нової ітерації того циклу, в якому цей виклик здійснено.

Приклад 3.7

Як і у прикладі 3.4, розв'яжемо задачу перевірки на простоту натурального числа *n,* застосувавши той самий алгоритм. Але у програмній реалізації алгоритму відмовимося від логічної змінної Тіад, перериваючи виконання циклу в разі знаходження дільника *n* оператором виклику процедури Ьгеак. Після завершення циклу саме значення потенційного дільника *к* визначатиме, чи є число складеним: якщо к=Тгипс($ягТ(п))+1, то оператор виклику процедури Ьгеак не було застосовано жодного разу і число *n* є простим, інакше вихід із циклу було здійснено оператором виклику процедури Ьгеак внаслідок того, що знайшовся дільник *n,* а отже, число п є складеним. Результати роботи програми подано на рис. 3.12.

```
ргодгат ех3_7;                          {визначити прості числа}
уаг п,к:іпТедег;                        {число, поточний дільник}
Ьеді п
   мгіТе!п('пишЬег зішрі ісіТу');
   мгіТе1п('епТег іпТедег:  ');
   геасііп(п);                                  {увести число}
   к:=2:                                {вибрати перший дільник}
                              {перебирати потенційні дільники}
   мНїїе (к<=Тгипс(здгТ(п))) сіо
   Ьеді п
     іТ п тосі к =0 ТІїеп                  {якщо к ділить п,}
       Ьгеак;                            {сигналізуємо про це}
     к:=к+1;                              {наступний дільник}
   ЄПСІ;
   ІТ к=Тгипс(5дгТ(п))+1 ТІїеп мгііе1п(п.' із вітріє')
                         еізе и/гіїе1п(п,' із поТ зішріе');
єпсі.
```

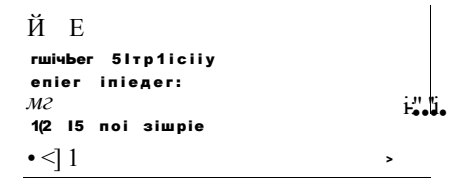


Рис. 3.12. Результати роботи програми ех3_7. Визначення простих чисел

3.3. Деякі циклічні алгоритми та програми

Одним із найпростіших і найважливіших застосувань циклічних структур є генерування рекурентних послідовностей. Ефективність розв'язання деяких математичних задач цілком залежить від вибору рекурентної послідовності та способу її

обчислення. До таких задач належать, зокрема, задачі обчислення значень степеневих рядів і трансцендентних функцій, що розглядаються у розділах 3.3.2 та 3.3.3 відповідно. У розділі 3.3.1 дається означення рекурентної послідовності і розглядається загальний алгоритм її генерування.

3.3.1. Рекурентні послідовності та співвідношення

Під час обчислення факторіала за програмою ex3_6 багаторазово застосовується формула г! =(г-1)! - *i,* реалізована оператором `Tacioriai :=Tacïoria1*i`. Наведемо значення лічильника циклу i та відповідні значення змінної `TacToriaI`.

i	TacToriaI
1	1
2	2! = 1! • 2 = 2
3	3! =2! • 3 = 6
4	4! =3! • 4 = 24

Отже, змінна `TacToriaI` послідовно набуває значень 1, 2, 6, 24,.... Позначивши члени цієї ПОСЛІДОВНОСТІ через *PO, Pb P2,* •••> *Pm* ОТРИМАЄМО рівність: *PI = PI_i* • *i,* де i = 1, 2, ..., *n.* Така рівність є прикладом рекурентного співвідношення. Дамо означення цього поняття.

Формула, що виражає член послідовності через один або декілька попередніх, називається *рекурентним співвідношенням.* Послідовність, члени якої задовольняють деякому рекурентному співвідношенню, називається *рекурентною.*

Найпростішими прикладами рекурентних послідовностей є арифметична та геометрична прогресії, елементи яких пов'язані з попередніми елементами співвідношеннями $a_n = a_{n\_x} + ci$ та $a_n =$ • д, де *ci* та ^ — деякі сталі величини. Іншим відомим прикладом рекурентної послідовності є послідовність чисел Фібоначчі, яку розглянемо детальніше.

Приклад 3.8

Послідовність {/)} чисел 1,1, 2, 3, 5, 8,13,..., де /i =/2= 1, а кожний наступний член дорівнює сумі двох попередніх, називається послідовністю чисел Фібоначчі. Таким чином, усі члени послідовності чисел Фібоначчі, починаючи з третього, задаються рекурентним співвідношенням /„ = /„_2 + /п_i. Числа Фібоначчі мають декілька цікавих властивостей. Зокрема, сусідні числа Фібоначчі є взаємно простими; найбільшим спільним дільником двох чисел Фібоначчі є число Фібоначчі; число Фібоначчі парне тоді і тільки тоді, коли його номер кратний трьом. Зауважимо, що числа Фібоначчі використовуються при побудові ефективних алгоритмів розв'язання багатьох обчислювальних задач, таких як пошук екстремуму функцій, упорядкування даних.

Розробимо програму, що генерує послідовність *n* перших чисел Фібоначчі. Нам знадобиться змінна `Ti` для зберігання значення поточного члена послідовності та змінні `TI` і `T2` для зберігання значень двох попередніх членів. Значення `Ti` модифікуватиметься оператором `Ti :=T1+T2`, який увійде до складу циклу з лічильником.

Після виконання цього оператора слід переприсвоїти значення змінних TI та T2 так, щоб вони містили нову пару чисел Фібоначчі. Це можна здійснити операторами T1:=T2 та T2:=Ti. Порядок виконання останніх двох присвоювань є суттєвим; у разі виконання цих операторів у зворотній послідовності змінні TI та T2 містили б значення того самого числа Фібоначчі. Результати роботи програми, що генерує послідовність чисел Фібоначчі, подано на рис. 3.13.

```
ргодгат ех3_8;                              {послідовність чисел Фібоначчі}
уаг TI.T2,Ti:іпТедег;     {два попередніх та поточний член послідовності}
    i.niiniедег:    {лічильник і загальна кількість членів послідовності}
Ьеді п
  мгіТеСЕпТег ТНе ІепдТЬ оТ РіЬопассі зедиепсе: ');
  геасії п (п);                                        {ввести кількість членів}
  T1:=1;  T2:=1;                              {і ні ці алі зувати два перших члени}
                                                 {якщо послідовність містить}
  іТ п>0 ТНеп мгіТе(TI);                                {принаймні один член}
  іТ п>1 ТНеп мгіТе(' ',T2);                            {принаймні два члени}
  Тог і:=3 То п сіо                          {цикл обчислення наступних членів}
  Ьеді п
    Ti  =T1+T2;                          {обчислити поточний член послідовності}
    TI  =T2;                                    {сформувати нову пару доданків}
    T2  =Ti
    мгіТе(     .Ti);                          {вивести чергове число Фібоначчі)
  єпсі;
єпсі.
```

```
;ЕпІег £Ье Іепдїіі оГ РіЬопассі 5е9иепсе: 15
 1  1  2  3  5  8  13  21  34  55  89  144  233  377  610
```

ї й

Рис. 3.13. Результати роботи програми ех3_8. Обчислення послідовності чисел Фібоначчі

У загальному випадку рекурентне співвідношення визначає залежність члена послідовності *{5п}* від *к* попередніх: 5„ = *Р(5*п-ь,..., Таке число *к* називається *порядком рекурентного співвідношення.* Поширимо реалізовану в програмі ех3_8 схему обчислень на рекурентні співвідношення довільного порядку *к.*

Для обчислення чергового члена послідовності потрібні *к* попередніх членів, які зберігатимемо у змінних TI,..., Тк. Одразу зазначимо: значення ..., $5_к$ повинні бути задані у вигляді констант або введені з клавіатури. Тому змінні TI,..., Тк можуть бути ініціалізовані на початку програми серією присвоювань TI := 51,..., Тк := 5к, де 51,..., 5к вважатимемо константами. Члени послідовності, індекси яких більші за *к,* обчислюватимуться та виводитимуться в тілі основного циклу програми, а значення ^. . . . $.5_к$ слід вивести до початку виконання цього циклу.

Припустимо, що треба отримати *п* членів послідовності. Член послідовності з індексом *і* має бути виведений лише в разі істинності умови *і* < *п.* Для членів з індексами і, більшими за *к,* виконання умови *і* < *п* гарантуватиметься оператором

циклу, а для перших к членів цю умову слід перевіряти в операторах розгалуження. Наведемо один з можливих варіантів серії таких операторів разом із операторами ініціалізації змінних TI, Тк.

```
TI:=51; ... Тк:=5к:
iT п > 0 TIїеп мгiTeCTI):
iT п > 1 їЬеп мгiTe(' ',T2):

iT п > к-1 Тйеп мгiTeC' '.Тк);
```

Тепер розглянемо будову циклу, що обчислює члени послідовності з індексами *к*+1,..., *n*. Значення нового члена присвоюватимемо змінній Ti: Ti :=P(T1. . . . .Тк), де P(...) - функція або вираз, що реалізує рекурентне співвідношення. Модифікувавши значення Ti, слід виконати зсув значень змінних TI,..., Тк: T1:=T2;T2:=T3;... Тк:=Ti; (знову звернімо увагу на порядок виконання присвоєнь). Після цього в тілі циклу залишиться лише вивести значення нового члена послідовності. Покажемо схематично, як виглядає основний цикл програми.

```
Тог i: =к+1 То п сіо
Ьеділ
    Ti :=P(T1. . . . .Тк):
    TI:=T2;T2:=T3:...Тк:=Ti;
    мгіїе(Ti);
єпсі;
```

3.3.2. Степеневі ряди

Розглянемо задачу наближеного обчислення значення функції y = зіп x. Цю функцію можна розвинути у степеневий ряд Тейлора

„3 5 (- Ї Ї " - 1 * 2 " - 1

Наближене значення суми ряду можна отримати або обмежуючись сумою перших n його членів, або обчислюючи суму *з наперед заданою точністю.* Формула загального члена даного ряду є достатньо простою, але використовувати її не раціонально, оскільки для кожного члена ряду треба обчислювати степінь і факторіал. Набагато вищої ефективності можна досягти, обчислюючи член ряду за допомогою рекурентного співвідношення. Для знаходження цього співвідношення зауважимо, що чисельник п-го члена ряду дорівнює чисельнику $(n-1)$-го, помноженому на $-X^i$, а знаменник - знаменнику $(n-1>$го члена, помноженому на величину $(2n-2)\cdot(2n-1)$ для $n = 2, 3. . . .$ Позначивши через a_i- значення г'-го доданка, отримаємо таке рекурентне співвідношення:

$$a_i = \frac{-x^2}{(2i-2)(2i-1)} \qquad i = 2, 3,$$

Справді, $a_1 = x, a_2 = a_x \cdot (-X^i) / (2 \cdot 3) = -x^3 / 3!$, $a_3 = a_2 \cdot (-x^2) /(4-5) = x^3/5!$ тощо. Отже, стає зрозумілим ефективний спосіб обчислення суми n перших членів

ряду. Тепер перейдемо до обчислення суми ряду з наперед заданою точністю. По-перше, зазначимо, що із заданою точністю може бути обчислена сума лише збіжного ряду, а довільний степеневий ряд має певну область збіжності (можливо, порожню), тобто збігається не за всіх, а лише за деяких значень *x* (ряд, що розглядається нами як приклад, збігається для будь-якого дійсного *x).* По-друге, простий спосіб перевірки точності часткової суми ряду існує не для всіх рядів. Такий спосіб існує, зокрема, для знакопереміжних рядів, абсолютні величини членів яких, починаючи з деякого номера, утворюють монотонно спадну послідовність. Для таких рядів сума всіх членів, починаючи від (и+1)-го, є меншою за модулем від и-го члена. Ряд Тейлора для функції *y* = зіп *x* задовольняє щойно розглянуті умови. Отже, для цього ряду

$$a_{,,} > \frac{X^{a}}{;=\text{лн}i}*$$

Якщо задана похибка є > 0, то визначити останній доданок можна з умови $\backslash a_{,,}| < є$, і тоді часткова сума $5_n = a^{\wedge} + a_2 + ... + a_n$, відрізнятиметься від значення зін.г не більше, ніж на величину є.

Приклад 3,9

Обчислимо наближене значення функції *y* = зіп *x,* використовуючи її розвинення у ряд Тейлора. У змінній ерз зберігатимемо значення похибки, а у змінній іієгп - значення поточного члена ряду. Рекурентне співвідношення реалізоване присвоюванням ііет:=і1;ет*(-х*х)/(і*(і+1)), де лічильник і дорівнює подвоєному номеру члена ряду, збільшуючись на 2 на кожній ітерації циклу. Обчислення тривають доти, доки модуль величини ііет перевищує ерз (модуль величини і ієні записується на мові РазсаІ як аЬз(іієгл)). При виведенні результату поруч із обчисленим у програмі значенням зіп *x* буде виведене значення, що повертається вбудованою функцією зіп(х) (рис. 3.14).

```
ргодгат ехЗ_9;                                  {обчислення функції зіп(х)}
уаг х.з,іїеш:геа1;                  {аргумент функції, сума та член ряду}
    і:іпїедег;                                                  {лічильник}
    ерз:геаі;                                                   {точність}
Ьеділ
  мгіїе!п('зіп(х)  саісиїаііоп');
  мгіїе!п('епіег Типсііоп агдишепі; х:');
  геасЛп(х);
  мгіїе1п( 'епіег гпізіаке');
  геасіїп(ерз);                                  {увести похибку розрахунків}
  з:=х;                                            {і ні ці алі зувати суму ряду}
  І1ет:=х;                                 {і ні ці алі зувати перший член ряду)
  і:=2;                                            {ініціалізувати лічильник}
  мИіІе аЬз(іІет)>ерз сіо {доки поточний член не задовольняє точності}
  Ьеділ                          {обчислювати поточний член і суму ряду}
    ііет:=і1;ет*(-х*х)/(і*(і+1));
    з:=з+і£ет;
    і :-1+2;
  епсі;
```

```
мгіїеігк '5=' ,5:6:7,' зіп(',x:3:3.')=',5in(x):6:7,
        [1] ergor='.abз(з-5і п(x)):6:7);
```

єпсі.

```
                                                          УЕІЯі
;5In(x) caicиїaiion                                        «її
ienier Рипсiion агдитепі x:
1
epier пізіаке
0.0001
5=0.8414718 5ІП(1 -000) = o.8414710 ergor-0. 0B00000 т м
<П                                                          i
```

Рис. 3.14. Результати роботи програми ex3_9. Обчислення значення функції y = зіп *x*

3.3.3. Ланцюгові дроби

Розглянуті у попередньому розділі степеневі ряди - один із інструментів обчислення значень математичних функцій, таких як зіп *x,* соз *x,* е*, 1п *x,* агс!§ *x* тощо. Іншим інструментом, що найчастіше використовується у вбудованих підпрограмах, є *ланцюгові*, або *неперервні, дроби* (про вбудовані, або стандартні, процедури та функції йтиметься в розділі 4.1.3).

Дамо означення. Вираз

$$Ь_0 + \cfrac{a_1}{Ь_{1+}\cfrac{a2}{Ь_г, + \cfrac{«з}{Ь_г, +\ldots}}}$$

називається *ланцюговим дробом.* Нескінченний ланцюговий дріб позначається так:

$$\text{o}\ \frac{a_1}{Ь_1}\ \frac{a_2}{Ь_2}\ \frac{a_3}{Ь_3}\ \ldots$$

а скінченний ланцюговий дріб із останнім знаменником $Ь_n$ - так:

$$\frac{a_1}{Ь_1}\ \frac{a_2}{Ь_2}\ \ldots\ \frac{a_n}{Ь_n}$$

Скорочений запис цих позначень є таким:

$$Ь_0; \frac{a_к}{} \quad \text{та} \quad . \frac{a_к}{}$$

Ланцюгові дроби обчислюються з кінця.

Розглянемо послідовність {г,} знаменників ланцюгового дробу $\frac{a_i}{}$

$$2_n = Ь_n, \quad 2_{n\_}! = Ь_n\_x + a_n/2_n, \quad 2_n\_2 = Ь_n\_2 + a_n\_y/2_{n\_i}, \ \ldots,$$
$$2\text{-}Ь_x + a_2/2_2, \quad 2_0 = Ь_0 + a_1/2_i.$$

Послідовність визначається рекурентним співвідношенням $2, = Ьi + a_{i+i} /г_{i+i}$ для всіх $i = n - 1, n - 2,..., 0$ і першим членом $2_{„} = Ь_{n}$, а значення $г_0$ дорівнює значенню дробу. Тому значення скінченного ланцюгового дробу можна обчислити за тим самим алгоритмом, що і значення членів рекурентної послідовності.

Приклад 3.10

Обчислити ланцюговий дріб вигляду

$$1 + \cfrac{1}{3 + \cfrac{}{\ddots + \cfrac{}{2n-1 + \cfrac{1}{2n+1}}}}$$

Для цього дробу $a_i = 1, Ь_I = 2i + 1, i = 1,..., n, Ь_0 = 1$. Змінну Ь використаємо для зберігання значень 2i + 1, а змінну $г$ — для зберігання значень знаменників. Значення змінної Ь має зменшуватися на 2 кожної ітерації циклу. Крім модифікації значення Ь, в тілі циклу треба обчислити чергове значення 2 за формулою рекурентного співвідношення: $2_{i-1} = Ь_i + 1 / 2_i$. Результати роботи програми, що обчислює ланцюговий дріб, зображено на рис. 3.15.

```
ргодгат ех3_10;                    {обчислення ланцюгового дробу}
уаг 2,Ь:геаі;          {поточний знаменник і доданок до нього}
    п:іпіедег;                       {кількість дробових рисок}
Ьеdіп
   мгіїеІпС 'сіїаіп ТгасТіопз');
   к/гіїе1п('епіег пигльег оі Тгасїіопз п:');
   геасііп(п);                  {увести кількість дробових рисок}
   Ь:=2*п+1;
   2:=Ь;                                    {останній знаменник}
   мМ1е Ь>1 сіо
    Ьеdіп
      Ь =Ь-2;                                  {модифікувати Ь}
      2 =Ь+1/г;                  {визначити поточний знаменник)
    єпсі
   мгіїєіпс 'з  .2:6:3):  {вивести значення ланцюгового дробу}
єпсі.
```

```
: піхі
cЛain PracIion5
enier пипьer <И Гraciionї п:
9
5- 1.313
```

РИС. 3.15. Результати роботи програми ех3_10. Обчислення ланцюгового дробу

Як уже зазначалося, ланцюгові дроби можна застосувати для швидкого обчислення значень деяких математичних функцій. Функції зображуються у вигляді нескінченного ланцюгового дробу, елементи якого a_1(a_2,... і $Ь_0$, $Ь_ь$ $Ь_2$,... залежать від *x*. Наведемо такі зображення для декількох функцій, залишивши як вправу для самостійного виконання програмну реалізацію відповідних обчислень.

$$^{1;}\ 1'\ 2\ '3'\ 2\ '5 \qquad 2\ '2и + 1"".$$

$$\text{Іпл:} = 0;\ \frac{x \bullet \bullet 1}{1}\ \frac{x\text{-}1}{}\ \frac{x\text{-}1}{}\ \frac{2(x\ \text{-}1)}{}\ \frac{2(x\ \text{-}1)}{}\ \frac{n(x\text{-}1)}{2n+1}\ \frac{n(x\text{-}1)}{}$$

$$= 0;\ \frac{X\ \text{-}X^2}{\Gamma\ 3\ '\ 5}\ \frac{\text{-}X^2}{2n+1'\ ''}$$

$$\text{arcі§}\ x \qquad \frac{x\ x^{o}}{\Gamma\ 3\ '\ 5}\ \frac{4x^{o}}{2n+\Gamma}\ \frac{I\Gamma^2X^2}{}$$

Висновки

+ Альтернативне розгалуження дає можливість виконавцеві алгоритму вибрати один із двох сценаріїв подальших дій залежно від того, чи виконується деяка умова. В мові РазсаІ альтернативні розгалуження реалізуються умовним оператором ІТ...І;НЄП...ЄІ5Є.

• Одноальтернативне розгалуження дає можливість виконати певні дії в разі істинності деякої умови і не виконувати жодних дій в разі її хибності. У мові РазсаІ одноальтернативне розгалуження реалізується скороченою формою умовного оператора: іТ...ІІіеп.

• Операторний блок, або складений оператор, - це послідовність операторів, що розпочинається ключовим словом Ьеді п та завершується ключовим словом єпсі (слова Ьеді п та єпсі інколи називаються операторними дужками). Блок сприймається як єдине ціле та може знаходитися в будь-якому місці програми, де синтаксисом мови припускається наявність оператора.

• Поліваріантне розгалуження є узагальненням альтернативного розгалуження. Цей тип розгалуження дозволяє здійснити вибір однієї з декількох алгоритмічних гілок залежно від значення певного виразу. В мові РазсаІ таку алгоритмічну конструкцію реалізовано оператором вибору сазе.

• Цикл - це послідовність дій, які повторюються. Оператори циклу задають повторення деяких дій доти, доки певна умова залишається істинною або доки умова не стане істинною. Оператори циклу мають заголовок і тіло. У заголовку записано умову продовження чи завершення циклу, а в тілі - блок операторів, виконання яких повторюється. Змінні, значення яких модифікуються в тілі циклу і впливають на істинність умови його завершення чи продовження,

називаються параметрами циклу. Кількість повторень циклу може бути наперед заданою або визначатися під час його виконання. Кожне повторення тіла циклу називається ітерацією.

- У мові РазсаІ є три різновиди операторів циклу: оператор циклу з передумовою, оператор циклу з постумовою та оператор циклу з лічильником. Вибір певного оператора циклу для розв'язання тої чи іншої алгоритмічної задачі здійснюється залежно від типу умови завершення чи умови продовження циклу.
- Оператор циклу з передумовою м(ii1e...cIo застосовують у тому випадку, коли кількість повторень невідома і цикл може не виконатись жодного разу. Умова продовження циклу перевіряється перед першим виконанням його тіла. До початку виконання циклу потрібно ініціалізувати його параметри, а оператор зміни поточних значень параметрів слід включити до тіла циклу.
- Оператор циклу з постумовою гереаТ...ипТі І застосовують тоді, коли кількість повторень невідома і цикл завжди виконується принаймні один раз. Умова завершення циклу перевіряється після виконання його тіла. До початку виконання циклу треба ініціалізувати його параметри, а оператор зміни поточних значень параметрів слід включити до тіла циклу.
- Оператори циклу з лічильником Тог...То...сіо або Тог...с!ошТо...с!о використовують тоді, коли кількість повторень є відомою до початку виконання циклу. Лічильник циклу є змінною деякого перелічуваного типу даних. Початкове значення лічильника має бути меншим за його кінцеве значення для оператора Тог...То...сіо і більшим — для оператора Тог...cIоиmТо...cIo.
- На кожній ітерації циклу Тог...То...сіо значення його лічильника збільшується на одиницю, а на кожній ітерації циклу Тог...сіонпТо...cIo - зменшується на одиницю. Робиться це автоматично.
- Формула, що виражає член числової послідовності через один або кілька попередніх членів, називається рекурентним співвідношенням. Послідовність, члени якої задовольняють певне рекурентне співвідношення, називається рекурентною.
- Обчислення математичних функцій зіп *x,* соз *x, є*,* 1п *x* і агсі§ *x* у вбудованих підпрограмах реалізується за допомогою ланцюгових дробів. Ланцюгові дроби обчислюють за допомогою рекурентного співвідношення.

Контрольні запитання та завдання

1. Як формується складена умова, що об'єднує декілька простих умов?
2. Наведіть синтаксис оператора одноальтернативного розгалуження.
3. До яких типів даних не може належати значення виразу-селектора в операторі вибору?
4. У чому полягає відмінність між циклами з передумовою та циклами з постумовою?

5. Якому типу даних може належати лічильник у циклі Tor?
6. Яке значення має лічильник після завершення циклу Tor?
7. Що може спричинити «зациклення» програми?
8. За яких умов цикли иііііе та Tor не виконаються жодного разу?
9. Коли цикл виконується лише один раз?
10. У чому полягає відмінність між такими операторами циклів, як TOr...TO...cI0 та Tor...сіомпTo...сіо?
11. Яка структура працює ефективніше: вкладені оператори і T...THеп...e1 зе чи серія операторів і T...TИеп? Відповідь обґрунтуйте.
12. Чи можна перервати роботу циклу, не використовуючи процедури Ьгеак?
13. Чи можна перервати роботу програми за допомогою процедури Ьгеак?
14. Що таке рекурентні співвідношення і де вони використовуються?
15. Визначте поняття ланцюгових дробів і наведіть приклади їх використання.
16. Чи можна пропустити деякі оператори програми, що не належать тілу циклу, за допомогою процедури сопTіпие?
17. Як **підвищити** ефективність роботи вкладених структур іT...TИеп...e1зе?
18. Наведіть приклади рекурентних співвідношень.

Вправи

1. Визначити значення змінних x, і та у під час виконання такої послідовності операторів:

```
1 :=i; x:=1; y:=2;
мНіїе x<y сіо
Ьеділ
  i:=i+1: x:=x*i; y:=y*2;
  и п Т е Ш і . ' ', x. ' ', y)
єпсі;
```

```
I :=1; x:=1; y:=2;
мTiiIe i <=10 сіо
Ьеділ
  i:=i+1: x:=x*i; y:=y*2;
  мгіTe1п(i, ' ', x, ' ', y)
єпсі;
```

2. Визначити значення змінних а та з під час виконання такої послідовності операторів:

```
а;=1; з:=1;
герєаТ
   а:=2*а: з:=з+а;
ипTii а>10;
мгіTe1п(а. з):
```

```
а :=1;
гереаї
   а:=а+2;  игіТеША);
ипТії  а=4;
```

3. Нехай 5 означає довільний оператор, *В* - довільний логічний вираз. Виразити:

   а) цикл мііііе *В* сіо 5 за допомогою гереаТ-циклу;

   б) цикл гереаї 5 ипііі *В* за допомогою мііііе-циклу.

   Допускається використання умовного оператора.

4. Нехай 5 та *Р* - довільні оператори, *В* — довільний логічний вираз. Виразити оператор іТ *В* Тьеп 5 еізе Р:

   а) за допомогою операторів циклу та виклику процедури Ьгеак;

   б) за допомогою операторів циклу, не викликаючи процедуру Ьгеак.

5. Яке зображення буде виведене в результаті виконання наведеного нижче фрагмента програми?

```
Тог і :=1 То 5 сіо
 Ьеділ
   Тог з:=1 То 3 сіо
     Ьеділ
       Тог к:=1 То 4 сіо
         мгіТе('*');
       мгіТеїп;
     єпсі;
     мгіТеїп;
 єпсі;
```

Задачі

1. Користувач вводить чотири числа: а, Ь, с, *ci.* Визначити, чи можна прямокутник зі сторонами *а* та *Ь* приклеїти до прямокутника зі сторонами с та *ci* так, щоб утворився новий прямокутник.
2. Обчислити значення функції соз х, використавши її розвинення у ряд Тейлора:

$$\text{соз } x = 1 - \text{лг}^2/2! + x^i/4! - x^6/6! + \ldots.$$

3. Обчислити значення квадратного кореня $y = 4\%$ за рекурентною формулою:

$$\text{Уі+I} \quad 2 \qquad \text{У}_I). \quad \text{i} = 0, 1, 2$$

4. Обчислити для заданого натурального числа *n* вираз:

$$\sqrt{3+\sqrt{6+\ldots+\sqrt{3(n-1)+\sqrt{3n}}}}.$$

5. Задане натуральне число *n.* Визначити кількість цифр у числі *n,* підрахувати їх суму та знайти першу і останню цифри.
6. Скласти програму обчислення значень функції $y = 4x^3 - їx^1 + 5$ для *x,* що змінюється від -3 до 1 з кроком 0,1.
7. Скласти програму, яка обчислює *n\\,* де *n* означає 1 x 3 x 5 x...x п, для непарного *n* та 2 x 4 x 6 x...x *n,* для парного *n.*
8. Задані дійсні числа *x* 0 та є > 0. Обчислити суму

$$\sum_{a=1}^{M}$$

з точністю до є. Визначити кількість доданків та вивести проміжні результати.

9. Для заданого а обчислити ланцюговий дріб

$$a^2 + \cfrac{1}{\cfrac{2}{«^2+-}}$$

$$\frac{a^2 + 256}{a}$$

10. Кожне число у трикутнику Паскаля, крім перших трьох, є сумою чисел, розташованих над ним у попередньому рядку (2 = 1+1, 3 = 1+2 тощо):

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

Вивести задану кількість верхніх рядків трикутника Паскаля.

11. Обчислити із заданою точністю константу п = 3,14159265 за допомогою ряду Лейбніца: я/4 = 1 - 1/3 +1/5 -1/7 + 1/9 - ....

12. Знайти всіх «близнюків», тобто пари простих чисел вигляду бп -1 і *6m* + 1 для т>0. Наприклад, 5 і 7, 11 і 13, 17 і 19 (але не 23 і 25). Розглядати числа, що не виходять за межі діапазону цілочислового типу даних.

Розділ 4

Процедурно-орієнтоване програмування

- **Означення процедур і функцій користувача**
- **Поняття формальних і фактичних параметрів, локальних і глобальних змінних**
- **Виклик процедур і функцій**
- **Способи передачі параметрів та їх повернення**
- **Програмний стек і процес виклику підпрограм**
- **Підпрограми як параметри**
- **Рекурсивні підпрограми, приклади ефективного та неефективного їх застосування**
- **Випереджальне оголошення процедур і функцій**

4.1. Підпрограми, їх різновиди та способи використання

Один з ефективних способів створення великих програм, *технологія низхідного проектування,* полягає в їх конструюванні за принципом «розділяй і пануй»: програма розглядається як набір маленьких фрагментів, кожний з яких виконує певну логічно завершену дію, може бути виконаний декілька разів та є більш керованим, ніж програма в цілому. Такий невеликий фрагмент програмного коду називається *підпрограмою.* Підпрограма позначається ідентифікатором, тобто має власне унікальне ім'я. Вказування в тексті програми імені підпрограми рівнозначне запису всіх її операторів і називається викликом підпрограми.

Отже підпрограма - це іменована частина програми, котра описує деякі обчислення і може бути викликана з будь якого місця програми, де синтаксисом мови це не заборонено. Таким чином, для багаторазового виконання деякого програмного коду достатньо записати його один раз у підпрограмі, і надалі, в разі потреби, вказувати лише її ім'я. Концепція програмування, що ґрунтується на використанні підпрограм як стандартних блоків для створення нових програм, отримала назву *повторне використання коду.* Мови, в яких реалізовано механізми використання підпрограм, називаються *процедурно-орієнтованими.* Мова РаscaI належить до таких мов.

Зазначимо, іцо конструювання програм з невеликих «будівельних блоків» сприяє їх більшій ясності та гнучкості, що приводить до підвищення їх ефективності, якості та надійності. Одна з найпомітніших переваг такого підходу полягає у простоті механізму внесення змін і виправлень у процедурно-орієнтовані програми. Пояснимо цю тезу на прикладі. Припустимо, що програма створює зображення моделі одягу. Якщо програма не є процедурно-орієнтованою, то кожен ґудзик малюватиметься окремими, «своїми» операторами, і для того щоб змінити форму всіх ґудзиків, доведеться послідовно змінювати спосіб відображення кожного з них. Якщо ж креслення ґудзика записати у вигляді підпрограми, то внесення змін до цієї підпрограми приведе до автоматичної зміни зображення всіх ґудзиків.

В мові РазсаІ існує два різновиди підпрограм: *процедури* та *функції.* Функція відрізняється від процедури тим, що *повертає* деяке значення в точку її виклику. Іншими словами, під час виклику функції її ім'я може інтерпретуватись як ім'я деякої змінної величини. В розділах 4.1.1 та 4.1.2 розглядатимуться способи створення та використання процедур і функцій. Розділ 4.1.3 присвячено стандартним процедурам і функціям, які є частиною системи програмування РазсаІ. У розділах 4.1.4 - 4.1.6 підпрограми розглядаються в розрізі процесів, що відбуваються в оперативній пам'яті під час їх виконання. Нарешті, в розділі 4.1.7 описано процедурні типи — один із найбільш потужних інструментів процедурно-орієнтованого програмування.

4.1.1. Процедури користувача

Із прикладами процедур і функцій ми вже зустрічалися - це стандартні процедури введення-виведення геасі, игіїе, математичні функції, наприклад зіп(х), ігипс(х) тощо. Вони називаються стандартними тому, що є невід'ємною частиною системи РазсаІ. Наявність великої бібліотеки таких програмних засобів суттєво полегшує розроблення програм. Але у більшості випадків специфічні для певної програми дії не мають прямих відповідників у бібліотеках РазсаІ і для їх виконання програміст має створити власні процедури або функції, тобто *процедури* або *функції користувача.*

І процедури, і функції можна поділити на класи *підпрограм із параметрами* та *підпрограм без параметрів. Параметрами* називаються змінні, за допомогою яких відбувається передача даних до підпрограми, що викликається, із програмного блоку, що здійснює виклик. Отже, підпрограми без параметрів можуть бути незалежними від зовнішніх даних, а підпрограми з параметрами обов'язково їх використовують. У цьому розділі спочатку розглянемо процедури без параметрів, а згодом — процедури з параметрами.

Процедури без параметрів

Приклад 4.1_

Розробимо калькулятор нарахувань за депозитними внесками. Вхідними даними є сума внеску, *сіерозіі,* відсоткова ставка річного прибутку, *гаіе,* та термін дії рахунку, *регіосі*, що дорівнює цілому числу років. Калькулятор має виконувати дві

операції: розраховувати суму коштів на депозитному рахунку станом на кінець терміну його дії та розраховувати суми коштів станом на кінець кожного року протягом терміну дії рахунку. Для визначення кінцевої суми застосовується формула складних відсотків *зит = сіерозії* • (1 + *гаіе*)^{регіоА}а для розрахунку щорічних сум краще застосувати рекурентне співвідношення 5«те,- = *зит^і* • (1 + *гаіе).* Тут зипПі - сума коштів на рахунку станом на кінець г'-го року, *і* = 1,2, ..., *регіоді, зитп$_0$ = сіерозії.* Користувачеві потрібно надати можливість обчислювати прибуток за різними вхідними даними, не завершуючи роботи програми.

Застосовуючи технологію низхідного проектування, виділимо прості логічно завершені дії і покладемо їх виконання на процедури. Такими діями є: введення вхідних даних (процедура Іпіі), обчислення щорічних нарахувань (процедура Зої іЛі оп), обчислення кінцевої суми на рахунку (процедура Рі паї)і виведення даних на екран (процедура Вгоизе). Тепер можна розробляти основну програму, оперуючи ідентифікаторами Іпії, Зоїііоп, Вгомзе та Рірагї і не замислюючись над внутрішньою структурою відповідних підпрограм. Отже, щойно було побудовано загальну структуру процедурної програми, яка подана на рис. 4.1, *а.* На рис. 4.1, *б* схематично зображено взаємодію основної програми з процедурою Іпії.

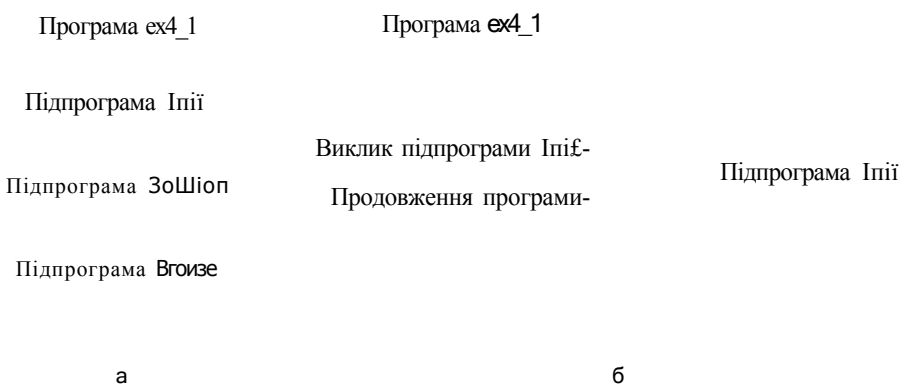


Рис. 4.1. Структура процедурної програми (а); взаємодія основної програми і процедури (б)

Побудуємо основну програму. Оскільки користувачеві потрібно надати можливість обчислювати суми нарахувань за різними вхідними даними, то процес введення даних та розрахунку результатів слід «зациклити», продовжуючи його доти, доки цього бажає користувач. На різних ітераціях такого циклічного процесу користувач може виконувати різні дії: введення даних, обчислення щорічних нарахувань або обчислення кінцевої суми. Вибір бажаної дії було б зручно здійснювати натисненням клавіші на клавіатурі. Для збереження значення вибраного пункту меню використаємо змінну кеу символьного типу, а вибір відповідної процедури здійснимо оператором вибору сазе із селектором кеу. Значення до змінної кеу вводитимемо за допомогою функції геасікеу, а не за допомогою оператора геасі, оскільки функція геасікеу не потребує натискання клавіші Епіег після введення

символу. Умовою завершення програми може стати натискання користувачем клавіші **Esc,** що приведе до присвоєння змінній key символу з кодом 27.

Зауважимо, що в результаті роботи калькулятора вікно програми може заповнитись даними розрахунків і тому варто додати до меню таку дію, як знищення результатів попередніх обчислень. Цю дію можна виконати стандартною процедурою сі гзсг. Проте саме меню під час очищення вікна програми не повинне зникати, тобто дію процедури Сігзсг треба обмежити тією областю вікна, у якій виводитимуться результати розрахунків. Таке обмеження здійснює процедура мі псіои. Наведемо код основної програми.

```
ргодгат ех4_1:
изез сгї;      {модуль сгі містить процедури сігзсг і мі псіом
                та функцію геасікеу}
уаг кеу:сіїаг;                                {вибраний пункт меню}

{================  основна програма  ==================}
Ьеділ
  сігзсг;
  мгі!е(' 1. епіег сіаТа    [1]);
  мпІе('2. уеаг Ьу уеаг зит    ');
  мгіїеІпС3. "ТІпаї зит'):
  мгі1:е('4. сіеаг саісиїаііопз    ');
  мгіїе1п('Е5С - ехії');
  \лпїеІп('================================================');
  міпсіом(1.4,80,25); {виключення меню з робочої
                       області вікна програми}
  гереаї;
    мпІеІпС'сЬоозе сотшапсі (1-4 ог Е5С):');
    кеу :=геасікеу;
    сазе кеу оТ
      '1' | Іпіїї;                                {введення даних}
      '2' | Боїиііоп:       {обчислення щорічних сум на рахунку}
      '3' | РІпаї;                    {обчислення кінцевої суми}
      '4' | СІ ГЗСГ;   {очищення робочої області вікна програми}
    єпсі;
  ипТі! кеу=#27; {#27 - код клавіші Esc}
єпсі.
```

Під час компіляції програми виникне синтаксична помилка ШедаІ аззідптепі (Недопустиме присвоєння), яка свідчитиме про те, що програмі невідомі ідентифікатори Іпіїї, Зоїиїіоп, Вгомзе, Рі паї. Очевидно, ці імена потрібно оголосити до їх використання. Але на відміну від імен змінних згадані імена є назвами процедур, тобто груп операторів, що виконують певні дії. Тому для їх оголошення використовуються інші синтаксичні конструкції. Оголошення процедури або функції записується в розділі оголошень програми і має таку саму структуру, як і вся програма, за винятком крапки наприкінці, замість якої використовується крапка з комою. Синтаксис оголошення процедури є таким:

```
ргосесііге <ім'я>(<оголошення параметрів>);
оголошення імен>
```

```
Ьеdіп
  <операторна частина процедури>
єпсі,
```

Тут ргосесіиге, Ьеді п, єпсі — зарезервовані слова; <і м' я> - унікальний в межах програми ідентифікатор процедури, за яким вона викликається; оголошення параметрі в> - оголошення змінних, яким надаються значення під час виклику процедури; оголошення і мен> — розділ оголошень ідентифікаторів, що будуть використовуватись лише в межах процедури; операторна частина процедури> - оператори, які виконуються під час виклику процедури, їх називають також *тілом процедури.*

Рядок програми, що містить ключове слово ргосесіиге, ім'я процедури та її параметри, називається *заголовком процедури.* Процедури можуть не мати параметрів, тоді вони називаються *процедурами без параметрів,* а синтаксис їх оголошення є таким:

```
ргосесіиге <ім'я>:
оголошення імен>
Ьеdіп
  операторна частина процедури>
єпсі:
```

Виклик процедури здійснюється за її іменем, яке використовується в основній програмі або в інших процедурах як окремий оператор. Отже, формат оператора виклику процедури без параметрів такий:

```
<ім'я процедури>;
```

Продовжимо низхідне проектування калькулятора нарахувань за депозитними внесками. Для того щоб позбутися синтаксичних помилок, слід ті підпрограми, методи реалізації яких ще не визначені, замінити «заглушками», тобто процедурами, які не виконують жодних дій. Зокрема, програма калькулятора успішно компілюватиметься, якщо перед словом Ьеdіп записати такі «заглушки»:

```
ргосесіиге Іпіі;
Ьеdіп
єпсі;

ргосесіиге Вгомзе;
Ьеdіп
єпсі:

ргосесіиге Зоїиііоп:
Ьеdіп
єпсі;

ргосесіиге Ріпаї:
Ьеdіп
єпсі;
```

Наступний етап розробки програми полягає у кодуванні дій, які мають виконувати процедури. Для реалізації процедури Іпіі потрібно оголосити змінні, що зберігатимуть значення вхідних даних. Нехай це будуть змінні сіерозіі (початкова

сума внеску), регіосі (термін дії рахунку) та гаіе (відсоткова ставка річного прибутку). Сама будова процедури Іпіі є досить очевидною, тому не потребує коментарів.

У процедурі обчислення щорічних нарахувань Зоїиїіоп міститиметься цикл, у якому регіосі разів застосовуватиметься рекурентна формула зит:=зит*(1+гаіе). Отже, у змінній зит зберігатиметься поточна сума коштів на депозитному рахунку. Оскільки процедура Зоїиііоп має виводити суму коштів станом на кінець кожного року, то в тілі згаданого циклу повинна викликатися процедура Вгомзе. Якщо номер поточного року зберігати у змінній уеаг, дія процедури Вгомзе обмежиться виведенням значень змінних уеаг та зит, і її будова також є очевидною.

Нарешті, розглянемо процедуру Ріпаї. Основна дія цієї процедури - обчислення кінцевої суми за формулою *зит - сіерозіі • (1 + гаіе)*^{регюсі}, що на мові РазсаІ записується так: зит:=сІеро5іі*ехр(регюсі*1п(1+гаіе)) (математичні функції та спосіб обчислення степенів детально розглянуто в розділі 4.1.3). Після обчислення значення змінної зит процедура Ріпаї має викликати процедуру Вгомзе. Але, оскільки процедура Вгомзе оперує значенням змінної уеаг, цій змінній у процедурі Ріпаї повинно бути присвоєне значення змінної регіосі.

Тепер можна зібрати всю програму, скомпілювати її та виконати. Нижче наведено повний код програми калькулятора нарахувань за депозитними внесками, а результати її роботи зображені на рис. 4.2.

```
ргодгат ех4_1; {калькулятор нарахувань за депозитними внесками}
изе5 сгі;
уаг кеу:сііаг;
    сіерозіі:геаі:                          {початковий внесок}
    р. . іосі: іпіедег;                     {термін дії рахунку}
    гаіе:геа1:          {відсоткова ставка річного прибутку}
    зиш:геа1;                               {сума на депозиті}
    уеаг:іпіедег;                               {поточний рік}
{=============== ввєДєння даних ================}
ргосесіиге Іпіі;
Ьеділ
  мгііе!п('епіег сіерозіі ');
  геасіїпССІерозіі):
  мгііеІпС'епіег регіосі ');
  геасіїп(регіосі);
  мгііе!п('епіег гаіе:5..20 ');
  геасії п(гаіе);
  гаіе:=гаіе/100:         {перетворення цілочислового відсотка
                                          на коефіцієнт гаіе}
єпсі;
{======= виведення результатів розрахунків ========}
ргосесіиге Вгомзе;
Ьеділ
  мгііеІпСзит аі іЬе єпсі оі '.уеаг,' уеаг із ',зит:6:2);
  мгііе1п('ргезз епіег іо сопііпие');
  геасПп;
єпсі;
```

```
        обчислення щорічних сум на рахунку
procesiиre Zoїиiion;
Ьediп
  зиіл:-ciерозіі:                          {початкове значення зит}
  Tor year:=1 To periosi cio
    Ьediп
      зит:=зиш*(1+raTe);                   {рекурентна формула}
      Ьromze;
    єпсі;
єпсі;
procesiиre Piпaї;
Ьediп
  year :=peri osi;
  {розрахунок зит за формулою складних відсотків}
  зит:=c!ерозіT*exp(1-п(1+raTe)*periosc1):
  Bromze;
єпсі;
|==============  основна програма  ================}
Bediп
  сігзсг:
  мгіTe('1. enTer ciaTa     ');
  мгile('2. year Ьy year зит     ');
  мгіїeIпC3. "Tiпaї зит');
  мгile('4. ciear caicиїaiionz     ');
  мгіїeIпCE3C - exiї');
  МГПeIПС'==============================================');
  міпс1ом(1,4,80,25);
  гереаї
    мгile1п('cHoozе соттапсі (1-4 or E3C):');
    кеу:-geacікеу;
    сазе кеу оТ
      '1':IпiT;
      '2':5o1иTion:
      '3':PIпaї;
      '4':CIГ5CГ;
    єпсі;
  ипііі кеу=#27;
єпсі.
```

```
C:\BP\BIN\EX4_1.EXE
:1. enier йaia     2. year Ьy year 5иїч     3. Гiпaї 5ии
    ciear ca1си1зCiол5     E3C - exii

CIЮOБЄ соїлпапсі (1-4 or E5C):
enier Йерозіі:
1000
enier perioci
18
enier rale:5..20
10
CIШ05Є соїптапй (1-4 or E3C):
5ип ai IИe епй o? 10 year I5 2593.74
pre55 enier Io coniinие
```

РИС. 4.2. Результати роботи програми ex4_1. Калькулятор нарахувань за депозитними внесками

Процедури з параметрами

У програмі ex4_1 відбувався обмін даними між процедурами. Зокрема, процедури ЗоїиТіоп та Ріпаї «передавали» процедурі Вгомзе значення змінних зит та year. Така передача даних дещо порушувала принципи низхідного проектування, оскільки процедури Зоїиііоп та Ріпаї повинні були «знати» внутрішню структуру процедури Вгомзе. В ідеальній ситуації загальновідомим є лише заголовок підпрограми, а її внутрішня будова залишається прихованою від розробників інших підпрограм, програм або модулів. Як за такої ситуації слід модифікувати значення змінних, що використовуються всередині підпрограм? Для цього застосовуються спеціальні змінні, *параметри процедури,* що оголошуються в її заголовку. Параметри відіграють роль своєрідного буферу між процедурою та «зовнішнім світом»: бажано, щоб усі значення, які надходять до процедури ззовні, були присвоєні її параметрам.

В оголошенні підпрограми імена параметрів записуються в круглих дужках після її назви. Наприклад, змінна х є параметром підпрограми зіп(х), що обчислює синус числа х. Під час виклику підпрограми, що має параметри, поруч із її іменем в круглих дужках записуються певні вирази. Значення цих виразів присвоюватимуться параметрам підпрограми. Наприклад, під час виклику зіп(3+2) підпрограми зіп(х) змінна х набуде значення 5. Значення параметрів підпрограми, що вказуються під час її виклику, називаються *аргументами* підпрограми.

Розглянемо приклад, що демонструє використання процедур з параметрами.

Приклад 4.2

Знайдемо просте число за його номером у послідовності всіх простих чисел. Першими членами цієї послідовності є числа 1, 2, 3, 5, 7, 11. . . .Тому третім за номером простим числом є 3, а п'ятим - 7 тощо. Наведемо спочатку алгоритм розв'язання задачі, а згодом, скориставшись технологією низхідного проектування, розробимо програму.

Алгоритм пошуку простого числа за його номером

1. Увести номер простого числа.
2. Покласти початкове значення лічильника простих чисел рівним одиниці.
3. Перше число, що перевіряється, покласти рівним одиниці.
4. Поки лічильник простих чисел не досягне введеного номера, повторювати дії:
   4.1. Перейти до перевірки наступного числа.
   4.2. Якщо число просте, збільшити лічильник простих чисел.
5. Останнє просте число з тих, що переглядалися на кроці 4, і є шуканим.

Проектування програми почнемо з розробки її основної частини, що реалізує визначені алгоритмом дії. В алгоритмі використовуються такі змінні величини: номер простого числа, значення якого обчислює програма (питЬег), лічильник простих чисел (,]), число, що перевіряється на простоту (п), та ознака того, що число є простим (Тіад). Змінні питЬег,] та п є, очевидно, цілочисловими, а змінна Тіад - булевою. Власне перевірку числа на простоту залишимо процедурі ІзЗішрІе, пара-

метрами якої будуть змінні п та Лад. При цьому значення змінної Тіад процедура ІзЗішрІе змінюватиме. Отже, тіло основної програми матиме такий вигляд:

```
Веділ
  мг1їе1п('епТег пишЬег ');
  геасії п(пишЬег);                {увести номер простого числа}
  ,]:=1:                          {поточний номер простого числа}
  п:=1;           {початкове значення шуканого простого числа}
  мМІе 1<пишЬег сіо          {поки лічильник не досяг заданого}
  Ьеділ                          {номера, повторювати такі дії:}
    п:=п+1;                              {вибрати наступне число}
    Із5ішр1е(п,Т1ад);          {перевірити, чи є число простим}
    іТ Тіад Тіїеп                {якщо число просте, збільшити}
      1:=,]+1;                         {лічильник простих чисел}
  єпсі;
  мгіТе1п(^-',л.' сИі5Іо=',п);              {п - шукане число}
єпсі.
```

Перш ніж записати оголошення процедури ІзЗішрІе, наведемо повний синтаксис заголовка процедури з параметрами:

```
ргосесііиге <ім'я>(<ім'я:тип>;... <уаг ім'я:тип>;...);
```

Парою <ім'я:тип> позначається ідентифікатор параметра та ідентифікатор його типу. Зазначимо, що параметри одного типу можна об'єднувати в один список: <ім'я1, ім'я2:тип>. Зарезервоване слово уаг записується перед іменем параметра процедури в тому разі, якщо процедура повинна модифікувати значення змінних, оголошених поза її межами. Так, процедура ІзЗішрІе повинна змінювати значення зовнішньої змінної Тіад. Ім'я такої змінної під час виклику процедури має відповідати параметру, оголошеному зі специфікатором уаг. Усі модифікації уаг-иараметра в тілі процедури відбиватимуться і на змінній, що була вказана у виклику процедури як значення цього параметра. Отже, заголовок процедури ІзЗішрІе може бути таким:

```
ргосесііиге Із5ітр1е(а; іпТедег; уаг Ь: Ьооїеап);
```

Перевірка числа на простоту в процедурі ІзЗішрІе виконується так само, як і у програмі ех3_4. Під час виклику процедури аргумент п надає значення параметру а, аргументом Тіад заміщується параметр Ь. Оскільки перед параметром Ь в оголошенні процедури вказане слово уаг, всі модифікації значення змінної Ь відбуватимуться і зі значенням змінної Тіад.

Докладніше параметри підпрограм розглядатимуться у розділі 4.1.4, а зараз наведемо повний код програми обчислення простого числа. Результати роботи програми зображено на рис. 4.3.

```
ргодгат ех4_2:          {пошук простого числа за його номером}
уаг пишЬег,                    {номер шуканого простого числа}
                                    {лічильник простих чисел}
    п:іпТедег;           {число, що перевіряється на простоту}
    Т1ад:Ьоо1еап;            {ознака того, що число є простим}
```

```
procesciire Iз5iшр1е(a:iniедег; уаг Ь:Ьоо1еап);
       {а - число, що перевіряється на простоту,
        Ь - ознака того, що число просте}
   уаг к:iniедег:                          {потенційний дільник}
Ьеді п
   к:=2;                                   {вибрати перший дільник}
   Ь:=ігие:            {ще немає підстав вважати а складеним}
   {перебирати потенційні дільники}
   мМIе (к<=ігипс(зягі(а))) апсі Ь сіо
   Ьеді п
     iT п тосі к =0 iiieп                  {якщо к ділить п,}
       Ь:=Та1зе:                           {сигналізуємо про це}
     к:=к+1:                   {наступний потенційний дільник}
   єпсі;
єпсі:

Ьеді п
  мгііе!п('епіег пишьег ');
  геасПп (питЬег);
  3:=1;
  п:=1;
  міііie ,і<пштЬег сіо
  Ьеді п
    п:=п+1;
    Із5iшр1е(п,і1ад);
    iT Тіад Тїїеп

  єпсі;
  мгіТе!п('1=',з,'  ргіте=',п):
єпсі.
```

```
епіег питьег
20
}=20 ргіте=67
                                          Тп
а ї і                                      З
```

РИС. 4.3. Результати роботи програми ех4_2. Пошук простого числа за його номером

4.1.2. Функції користувача

Окрім підпрограм-процедур у мові РазсаІ використовуються підпрограми-функції. В математиці за допомогою функцій задають залежності між змінними-аргументами та змінними-значеннями функції. Ці залежності можна задавати аналітичним, графічним, табличним та іншими способами. В алгоритмічних мовах розглядаються тільки ті функції, для яких можна задати алгоритм обчислення їх значень. Програмний опис певного алгоритму обчислення значень називається *функцією.* Ім'ям функції, використаним в тексті програми, позначається *виклик функції.* Як уже зазначалося, функція відрізняється від процедури тим, що *повертає* деяке

значення в точку її виклику, тобто під час виклику функції її ім'я може інтерпретуватись як ім'я деякої змінної величини. А отже, функцію, на відміну від процедури, можна викликати у виразах. Наприклад, вираз зіп(5)+1 є коректним у тому разі, коли $іп(х) - функція, і некоректним, якщо зіп(х) - процедура.

Приклад 4.3

Обчислимо довжини сторін трикутника, заданого координатами вершин. Зрозуміло, що основною дією в цій задачі є обчислення відстані між двома точками на площині. Відстань *сі* між двома точками з координатами *(а.\, Ь{), (а2; Ь2)* визначається за формулою $ci = li(a_2 - a_x)^2 + (b_2 - b_x)$ Якщо задачу розв'язувати без використання функцій, то цю формулу доведеться запрограмувати тричі. Застосування функції, що обчислює відстань між двома точками, дасть можливість запрограмувати вищезгадану формулу лише один раз, але при цьому тричі здійснюватиметься виклик функції. Другий шлях є ефективнішим, адже запрограмувати виклик функції у загальному випадку значно легше, ніж переписати її операторну частину.

Наведемо синтаксис оголошення функції.

```
Типсііоп <ім'я>(оголошення параметрів>):<ім'я типу>;
оголошення імен>
Ьеділ
  операторна частина функції>
єпсі;
```

Тут Типсііоп, Ьеділ, єпсі - зарезервовані слова; <ім'я> - ідентифікатор функції, за яким здійснюється її виклик; оголошення параметрів> — необов'язковий список змінних та їх типів, синтаксис якого збігається із синтаксисом оголошення параметрів процедури; <ім'я типу> - тип значення, що повертається функцією; оголошення імен> — розділ оголошень ідентифікаторів, які використовуватимуться лише в межах функції; операторна частина функції> - оператори, що реалізують алгоритм обчислення значення функції, їх називають також *тілом функції.*

Ключове слово Типсііоп, ім'я функції, оголошення її параметрів і тип значення, що вона його повертає, становлять *заголовок функції.* Структура тіла функції повторює структуру програми: спочатку в тілі функції записуються оголошення ідентифікаторів, які в ній використовуються, а потім - оператори, що виконуються під час виклику функції.

УВАГА

Під час виконання функції останнім має виконуватись оператор присвоєння, в лівій частині якого записане ім'я функції. Значення, що присвоюється цим оператором, і вважатиметься значенням функції. Якщо внаслідок використання конструкцій розгалуження виконання функції може завершуватися різними операторами, то присвоєнь виразів імені функції має бути декілька.

Повернімося до задачі обчислення довжин сторін трикутника. Застосовуючи технологію низхідного проектування, визначимо структуру основної програми. Програма виконує три дії: введення координат вершин трикутника, обчислення довжин його сторін і виведення результатів.

Отже, тіло основної програми міститиме лише виклики трьох процедур:

```
ргодгат ех4_3;
Ьеділ
  Іпі"Ъ;                {введення координат вершин трикутника}
  Воіиііоп:              {обчислення довжин сторін трикутника}
  Вгомзе:                                {виведення результатів}
єпсі.
```

Визначимо змінні, через які відбуватиметься обмін даними між процедурами. Координати вершин зберігатимемо у змінних xi, yi, x2, y2, x3, y3, а довжини сторін - у змінних сїї, ci2, ЙЗ. Усі ці змінні належатимуть до дійсного типу даних.

Тепер перейдемо до проектування вищезгаданих процедур. Будова процедур Іпіі та Вгомзе є очевидною, а зміст процедури Зої иіі оп полягає у присвоєнні змінним сїї, С12 та сЗ значень, що дорівнюють відстаням між парами вершин. Ці відстані обчислюватимуться функцією Різіапсе. Процедура Зоїиїіоп матиме такий вигляд:

```
ргосесіиге Зоїиііоп:
Ьеділ
                                                  {відстані}
  СЇЇ =0ізіапсе(x1.y1,x2,y2)          {від (xi.yi) до (x2,y2)}
  ci2 =0ізіапсе(x2,y2,x3.y3)          {від (x2,y2) до (x3.y3)}
  cl3 =0ізіапсе(x1,y1.x3,y3)          {від (xi.yi) до (x3.y3)}
єпсі;
```

Операторна частина функції Оізіапсе міститиме лише один оператор: присвоєння імені функції значення, що розраховується за формулою визначення відстані між двома точками. Загальний вигляд оголошення функції Оізіапсе буде таким:

```
Типсііоп Оізіапсе(a1,Ь1,a2.Ь2:гєаі):геаі;
Ьеділ
  Оізіапсе :=5дгі(здг(а'і-а2)+5дг(Ь1-Ь2));
єпсі:
```

Наведемо **повний** код програми ех4_3 разом з оголошеннями процедур і функції (на рис. 4.4 зображено результати роботи цієї програми).

```
ргодгат ех4_3;          {обчислення довжин сторін трикутника}
уаг x1,y1,x2,y2,x3.y3:reai;     {координати вершин трикутника}
    cI1.ci2,ci3:reai;                {довжини сторін трикутника}
{===== обчислення відстані між двома точками =====}
Типсііоп 0ізіапсе(a1.b1.a2.b2:rea1):rea1:
Ьеділ
  Оізіапсе:=здгі(5дг(а1-а2)+5дг(Ь1-Ь2));
єпсі;
{===== введення координат вершин трикутника ====}
ргосесіиге Іпіі;
Ьеділ
  мгііе1п('епіег ігіапдіе арехез соогсііпаіез'):
  мгііе('coorciinaies xi.yi ')  геасіїп (xi.yi)
  мгііе('coorciinaies x2,y2 [1])  геасіїп (x2,y2)
  мгііе('coorciinaies x3.y3 ')  геасіїп (x3.y3)
єпсі;
```

```
{====== обчислення довжин сторін трикутника
procesiiге Зоїиііоп;
Ьедіп
  cI1:=OI5iапce(x1,y1,x2.y2);
  cI2:=0izіапce(x2.y2,x3.y3);
  cI3 :=0i5iапce(x1,y1, x3.y3);
єпсі:
{============= виведення даних ==========
procesiiге Вгомзе:
Ьедіп
  мгііе1п('1епдїЬ1='.  <11:6:2) ;
  мгііе!п(' 1епдііі2='   CI2:6:2);
  мгііе!п('1епдіИ3='   CI3:6:2);
єпсі;
                основна програма
Ьедіп
  Іпіі;
  Зоїиііоп;
  Вгомзе;
єпсі.
```

```
enieі* Iгіапдіе арехез соогйіпаїеа
coorйіпаїез x1,y1  B в
coorИіпаІЄ5 x2,y2 2 в
coorйіпаїез x3,y3 1 Ц
1епдді()1=   2.00
1епдІ(і2=   і).12
1епдІГІЗ=   К.12
```

Рис. 4.4. Результати роботи програми ex4_3. Обчислення довжин сторін трикутника за координатами його вершин

4.1.3. Стандартні процедури та функції

Стандартні, або *вбудовані, процедури* та *функції* входять до складу бібліотек мови програмування, вони викликаються без попереднього оголошення. Стандартна бібліотека мови РазсаІ містить широкий набір процедур і функцій для виконання типових математичних розрахунків, операцій введення-виведення, управління процесами, оброблення рядків і символів, роботи з файлами тощо. Деякі із вбудованих процедур та функцій будуть розглянуті нижче.

Математичні процедури та *функції* (табл. 4.1) реалізують найбільш поширені математичні операції. Тип аргументу функції може бути як цілим, так і дійсним. Більшість функцій повертає значення типу геаі, а в режимі компіляції з використанням сопроцесора або з його емуляцією - значення типу exіепсіесі. Деякі функції повертають значення, тип якого залежить від типу аргументу.

УВАГА

Значення математичних функцій обчислюються наближено і для деяких аргументів можуть відрізнятися від справжніх математичних значень.

Таблиця 4.1. Вбудовані математичні функції та процедури

Ім'я функції або процедури	Призначення	Тип значення, що повертається
АЬз(х), функція	Визначення абсолютної величини *х*	Збігається з типом аргументу
Агсіап(х), функція	Обчислення кута, тангенс якого дорівнює *х;* значення кута задане в радіанах	reai
Соз(х), функція	Обчислення косинуса *х;* параметр задає значення кута в радіанах	reai
Ехр(х), функція	Визначення експоненти *х,* тобто значення е*, де е = 2,718282 — основа натурального логарифма	reai
іп(х), функція	Обчислення натурального логарифма х	reai
5іп(х), функція	Обчислення зіп х; параметр задає значення кута в радіанах	reai
5дг(х), функція	Піднесення до квадрата значення *х*	Збігається з типом аргументу
ЗцгКх), функція	Обчислення квадратного кореня з *х*	reai
Рі, функція	Обчислення значення числа ті = 3,14159265...	reai
Іпі(х), функція	Обчислення цілої частини *х*	reai
Ргас(х), функція	Обчислення дробової частини *х.* Дробова частина розраховується за формулою: *{x)=x- [x]*	reai
Капсіоші ге, процедура	Ініціалізація вбудованого генератора псевдовипадкових чисел поточним системним часом	
гапсіош та гапсіот(х), функція	Генерація псевдовипадкових чисел. Якщо функція не має параметра, то повертає дійсний результат з діапазону 0...**1**, інакше — ціле число з діапазону 0..х - 1	reai або іпіедег

Зауважимо, що функції піднесення до степеня в мові РазсаІ немає. Значення степеня $a^x = (e^{\ln a})^* = e^{*\ln a}$ визначається виразом Exp(x*i.n(a)).

Далі розглянемо більш детальніше функції знаходження цілої та дробової частин числа. Нагадаємо, що ціла частина числа *х* - це найбільше ціле, що не перевищує *х*. Наприклад, [5] = 5; [-5] = -5; [5,8] = 5; [-5,8] = -6. Дробова частина числа - це різниця між самим числом і його цілою частиною. Наприклад, {5} = 5 - 5 = 0; {5,8} = 5,8 - 5 = 0,8; {-5,8} = -5,8 - (-6) = 0,2. Результати виконання функцій Іпі(х) і Ргас(х) збігаються з результатами виконання математичних операцій знаходження цілої та дробової частин тільки на множині невід'ємних чисел. Якщо число від'ємне, то функція Іпі(х) відкидає дробову частину, а функція Ргас(х) - цілу частину. Наприклад: Іпі(-5.8)= -5: Ргас(-5.8)= -0.8.

У табл. 4.2 наведено функції перетворення типів, що забезпечують отримання цілого числа з дійсного, символу за його кодом тощо.

Таблиця 4.2. Вбудовані функції перетворення типів

Ім'я функції	Призначення
СЬг(х)	Повертає символ з кодом *х* згідно з кодовою таблицею А5СІІ; результат має тип сИаг, параметр — тип Ьуіе
Огсі(х)	Повертає цілочисловий порядковий номер значення *х,* яке належить порядковому типу даних; дозволяє отримати код А5СІІ символу *х*
Коипсі(х)	Повертає значення *х,* заокруглене до найближчого цілого
Тгипс(х)	Повертає найближче ціле число, яке не більше за *х,* якщо *х* > 0, та не менше за *х,* якщо *х* < 0

Зазначимо, що функцію Огсі(х) можна також включити до категорії функцій обробки порядкових типів даних, що були розглянуті в розділі 2. Нагадаємо, що іншими функціями обробки порядкових типів даних є функції Ргесі(х) та Зисс(х), які повертають значення попереднього та наступного елементів порядкового типу даних відповідно.

Наступна категорія стандартних процедур і функцій (табл. 4.3) призначена для обробки цілочислових даних. Ці підпрограми використовуються для зміни параметрів циклу та визначення парності чисел.

Таблиця 4.3. Процедури та функції обробки цілочислових даних

Ім'я функції або процедури	Призначення
йесСх, сіх), процедура	Зменшує значення цілої величини *х* на *(іх;* якщо параметр *сіх* не заданий, то *х* зменшується на 1
ІпсСх, сіх), процедура	Збільшує значення цілої величини *х* на *(Ьс* якщо параметр *сіх* не заданий, то *х* збільшується на 1
Осісі(х), функція	Перевіряє число *х* на парність, повертає значення ігие, якщо *х* непарне

Для переривання роботи програми, затримки на деякий час її виконання або для переривання циклу використовуються вбудовані процедури керування порядком виконання програми (табл. 4.4).

Таблиця 4.4. Процедури керування порядком виконання програми

Ім'я процедури	Призначення
Вгеак	Вихід із циклу
Сопііпие	Пропуск операторів циклу, які записано після виклику процедури Сопїі пие; завершення чергової ітерації циклу та перехід до наступної
Ехі і	Вихід з поточної підпрограми в основну програму; якщо викликається з основної програми, то програма припиняє роботу
НаЩЕхіШхіе)	Припинення виконання програми
ОеІауШ	Затримка виконання програми на І мілісекунд

Інші вбудовані процедури та функції будуть розглянуті у відповідних тематичних розділах.

4.1,4. Локалізація імен

Кожний ідентифікатор у програмі характеризується *областю дії імені* або *областю видимості.* Область видимості ідентифікатора — це область програми, в якій можна посилатися на даний ідентифікатор. У мові РазсаІ припускається довільна послідовність і кількість розділів, в яких іменуються ті чи інші об'єкти. Такі розділи можуть належати до різних програмних блоків: процедур, функцій або самих програм. Область дії іменування поширюється від точки, де ідентифікатор було оголошено, до кінця блоку, в якому це оголошення відбулося.

Щойно був сформульований універсальний принцип визначення області видимості ідентифікатора. Застосуємо цей принцип для визначення областей дій імен, що оголошені в підпрограмах. Отже, процедури та функції можуть містити власні розділи оголошень констант, змінних, інших процедур і функцій. Усі ідентифікатори, оголошені всередині процедури або функції, є локалізованими в ній, тобто вони є невидимими зовні підпрограми. Такі ідентифікатори називаються *локальними.* Зокрема, локальними є будь-які параметри підпрограми. Мета локалізації змінних полягає в тому, щоб надати доступ лише до тих даних, які потрібні для формулювання задач, що їх розв'язують процедури або функції, і зробити в такий спосіб постановки обчислювальних задач незалежними від методів їх розв'язань.

Крім власних локальних імен всередині підпрограми видимими є й деякі ідентифікатори верхнього рівня, тобто ідентифікатори, що їх було оголошено у «зовнішніх» програмних блоках. Ці ідентифікатори називаються *глобальними.* Зазначимо, що з глобальних імен видимими є лише ті, оголошення яких розташовані до оголошення даної підпрограми.

Ім'я, локалізоване у підпрограмі, може збігатися з будь-яким глобальним іменем, оскільки цим іменам зіставляються різні ділянки оперативної пам'яті. У такому випадку локальне ім'я перекриває глобальне та робить його недоступним у підпрограмі. Імена локальних змінних можуть також збігатися з іменами локальних змінних інших підпрограм, такі змінні є абсолютно незалежними, хоч і мають однакові імена.

УВАГА

Різні об'єкти можна іменувати однаково, але в різних програмних блоках. В оголошеннях тієї самої підпрограми або програми усі імена мають бути різними.

Приклад 4.4_

Наведемо програму, в якій використовуються однакові імена для різних об'єктів. Результати роботи програми зображено на рис. 4.5.

```
ргодгат ех4_4:          {демонстрація областей дії ідентифікаторів}
уаг і:іпіедег:     {глобальна змінна. її область дії - уся програма}
{================= процедури ==================}
ргосесііге рі;
  ргосесііге р2;                  {внутрішня відносно рі процедура}
  уаг і:іпїедег:    {локальна змінна. її область дії - процедура р2}
```

```
  Ьеділ
    і:*»2;                          {і. що оголошена у процедурі р2}
    мгі1:е1п( 'Тгош р2:' ,і);
  єпсі;
Ьеділ
  і:=1;                                  {і. що оголошена в ех4_4)
  Р2;
  илїеІпС 'Тгош рі:' , і);
єпсі:
|============== основна програма ================}
Ьеділ
  илІе1п( 'сііздріау зсоре уагіаЬІез');
  і:=0;                                  {і, що оголошена в ех4_4}
  PI:
  мгі іеіп('Тгош ех4_4:',і);
єпсі.
```

```
У Ш І М Ш Щ т Ш т      - ¯ х
йізріау гсоре иаі-іаЬ1е5
Я-ті р2:2
Тгот р1:1                          " 'і
Ггоїл ех4_4:1
```

Рис. 4.5. Результати роботи програми ех4_4. Демонстрація областей видимості ідентифікаторів

У програмі ех4_4 глобальну змінну 1 ініціалізовано значенням 0. Оскільки в процедурі рі жодних змінних не оголошено, то використаний в її операторній частині ідентифікатор і посилатиметься саме на глобальну змінну і. Натомість, у процедурі р2 локальна змінна і перекриває глобальну і тому присвоєння ї :=2 жодним чином не впливає на значення глобальної змінної.

Процедура рі програми ех4_4 модифікує значення глобальної змінної і. Модифікація глобальних змінних у підпрограмі називається *побічним ефектом підпрограми.* Цей ефект вважається явним, якщо він виникає внаслідок виконання операції присвоювання, та неявним, якщо глобальне ім'я вказане у виклику підпрограми як значення параметра, оголошеного зі словом уаг. Модифікації глобальних змінних у підпрограмах слід уникати, оскільки її наслідки у загальному випадку передбачити важко.

4.1.5. Різновиди параметрів

Область оперативної пам'яті, що її використовує програма, поділяється на сегмент коду, сегмент даних та сегмент стеку. В *сегменті коду* (64 Кбайт) зберігаються команди програми, в *сегменті даних* (64 Кбайт) — значення глобальних змінних, а в *сегменті стеку* - значення локальних змінних і параметрів підпрограм.

Параметри, що їх імена вказані в заголовку підпрограми, називаються *формальними,* оскільки під час компіляції їх оголошення вони не прив'язуються до

жодного реального об'єкта. Натомість значення, що замінюють формальні параметри під час виклику підпрограм, називаються *фактичними параметрами* або *аргументами.* Під час виклику підпрограми між її фактичними та формальними параметрами встановлюється однозначна відповідність щодо кількості параметрів, їх типів та порядку згадування.

Формальні параметри поділяються на параметри-значення, параметри-змінні, параметри-константи та нетипізовані параметри-змінні.

Якщо параметр оголошено як *параметр-значення,* то під час виклику підпрограми обчислюється значення відповідного аргументу і копія отриманого результату передається підпрограмі. Зміна параметрів-значень усередині підпрограми не впливає на значення змінних, що могли бути вказані як аргументи підпрограми, оскільки змінюються їх копії.

Якщо параметр оголошено як *параметр-змінну,* то до підпрограми передається *покажчик на параметр,* тобто адреса певної змінної в сегменті даних оперативної пам'яті. Тому підпрограма виконує дії над значеннями параметрів-змінних, а не над їх копіями, і модифікація параметра-змінної приведе до модифікації змінної, що була вказана як аргумент в операторі виклику підпрограми. Параметри-змінні можна використовувати у процедурах з метою повернення отриманих під час виконання підпрограми значень до точки її виклику. У заголовку підпрограми перед іменем параметра-змінної необхідно записати зарезервоване слово var.

Для *параметра-константи* копія значення відповідного аргументу під час звернення до підпрограми не створюється. Значення такого параметра не можна змінювати у тілі підпрограми. Параметри-константи дозволяють зекономити оперативну пам'ять та підвищити швидкість виконання програми, оскільки їх використання зменшує витрати часу на виклик підпрограми. У заголовку підпрограми перед іменем параметра-константи треба записати зарезервоване слово const.

Якщо формальний параметр є *нетипізованим параметром-змінною,* то відповідний йому фактичний параметр може бути покажчиком на змінну довільного типу, тобто адресою сегмента даних оперативної пам'яті, де зберігаються значення, тип яких не відомий. У заголовку підпрограми перед іменем параметра-змінної слід записати слово var, але не треба вказувати тип параметра.

УВАГА

Немає потреби використовувати параметри-змінні в заголовках функцій, оскільки функція повертає значення за своїм іменем, а не за допомогою параметра.

4.1.6. Процес виклику підпрограми. Програмний стек

Нагадаємо, що під час виклику підпрограми їй передається керування. По завершенні роботи вона повертає керування програмі, що її викликала, у ту точку, з якої виклик було здійснено. Перша команда підпрограми називається *точкою входу,* а адреса такої команди — *адресою точки входу.* Оператор, що продовжує виконання програми по завершенні роботи підпрограми, називається *точкою повернення* із підпрограми. Точка повернення із процедури — це наступний за її викликом оператор або умова завершення чи продовження циклу, якщо виклик

процедури був останнім оператором в тілі циклу. Точкою повернення із функції може бути як оператор, так і окремий операнд певного виразу. У будь-якому разі точка повернення фізично є адресою в сегменті коду оперативної пам'яті.

Коли здійснюється виклик підпрограми, точка повернення з неї запам'ятовується і зберігається до завершення роботи цієї підпрограми. Для збереження точки повернення використовується ділянка оперативної пам'яті. Крім того, під час виклику підпрограми певні ділянки пам'яті зіставляються з її параметрами та локальними змінними. Сукупність усіх цих ділянок пам'яті називається *локальною пам'яттю* підпрограми. Якщо підпрограма є функцією, то до її локальної пам'яті додається ділянка для збереження значення, яке функція повертає. Ця ділянка ставиться у відповідність до імені функції.

Перш ніж розпочнеться виконання операторів тіла підпрограми, для неї виділяється локальна пам'ять і в цій пам'яті записується точка повернення. Потім обчислюються значення тих аргументів, що відповідають параметрам-значенням. Ці значення копіюються в локальну пам'ять, і таке копіювання називається *підстановкою аргументів за значенням.* Якщо процедура має параметр-змінну, то аргументом може бути лише ім'я змінної. Жодного обчислення її значення, тобто розіменування аргументу, не відбувається, а до локальної пам'яті записується адреса змінної-аргументу. Ця адреса ставиться у відповідність до параметра-змінної. Таке зіставлення називається *підстановкою аргументу за посиланням,* або *за адресою.*

Підсумовуючи сказане вище, сформулюємо алгоритм виклику підпрограми.

1. Для підпрограми виділяється локальна пам'ять.
2. В локальній пам'яті обчислюється й запам'ятовується точка повернення з підпрограми.
3. Обчислюються значення аргументів, що відповідають параметрам-значенням, і адреси аргументів, що відповідають параметрам-змінним. Здійснюється підстановка аргументів.
4. Виконуються оператори тіла підпрограми. Якщо підпрограма є функцією, то у локальній пам'яті запам'ятовується значення, яке функція повертає.
5. Здійснюється повернення з підпрограми. Якщо підпрограма є функцією, то з локальної пам'яті підпрограми до сегмента даних копіюється значення, яке функція повертає. Управління передається команді, що адресується точкою повернення з підпрограми.
6. Ділянка оперативної пам'яті, що була задіяна під локальну пам'ять, вважається вільною.

Виконання основної програми починається після завантаження її коду в оперативну пам'ять комп'ютера. При цьому відбувається виділення пам'яті для її змінних. Ці змінні доступні з програми протягом усього часу її виконання, і тому називаються *статичними.* Область пам'яті, що виділяється під програму та її змінні, також називається *статичною.* Із локальними іменами і параметрами-значеннями підпрограм, як правило, зіставляються ділянки іншої області пам'яті — *автоматичної.* Під час виконання викликів підпрограм пам'ять виділяється та звільняється автоматично, без явних вказівок у програмі.

Виділення та звільнення ділянок пам'яті під час виконання викликів підпрограм відбувається за принципом «останнім прийшов - першим пішов» (з англійської «Ьазі Іп — Рігзі Оиі» або скорочено ІЛРО). Якщо складати книжки в стопку і брати їх тільки зверху, то книжка, що потрапила у стопку останньою, забирається першою. Така стопка називається стеком (зГаск), а кладуть і забирають книжки за принципом ІЛРО. Тому автоматична пам'ять, що виділяється для підпрограм, програм називається ще *програмнім стеком.*

У програмному стеку, як і у стопці книжок, доступним є завжди тільки один елемент — верхній, що зветься *вершиною стеку.* Внутрішній стан стеку під час виконання програми однозначно визначається функціями 5зед, що повертає адресу сегмента стеку, та ЗрТг, яка повертає зсув покажчика вершини стеку в межах його сегмента. За замовчуванням розмір стеку дорівнює 16 Кбайт, його можна змінювати за допомогою директиви компілятора {$M} - Метогу АПосаТіоп Зігез 0**ігєс**т**іує.**

Приклад 4.5

На прикладі програми ех4_5 продемонструємо принцип використання різних типів параметрів підпрограм та роботу стеку.

```
ргодгат ех4_5:                                  {робота стеку}
уаг а, Ь;іпіедег;                             {глобальні змінні}
ТипсТіоп і(х:іпіедег):іпіедег;          {х - параметр-значення}
Ьеділ
  х:= х+1;
  Т;=х;
єпсі;
Типсііоп д(уаг х:іпТедег):іпТедег;        {х - параметр-змінна}
Ьеділ
  х:= х сііу 2;
  д:= х;
єпсі:
Ьеділ                                        {основна програма}
  а:= 12;
  Ь:= Т(д(а));                                 {виклик функцій}
  мгіТе1п('а='.а,'  Ь=',Ь);
єпсі.
```

Перед виконанням програми покажчики на функції, а також глобальні та локальні змінні ще не визначені. Як тільки програма починає виконуватися, визначаються покажчики на функції Т та д, що є адресами в сегменті коду. Це — точки входу до функцій. Цілочислові значення глобальних змінних на початку роботи програми ініціалізуються нулями, а перший оператор програми надає змінній а значення 12. Виконання оператора Ь:= Т(д(а)) починається з виклику функції д(а), або дС 12). Її адресу, як і адресу аргументу а, буде записано до стеку. Оскільки аргументом а заміщується параметр-змінна х функції д, то, коли функція д надає змінній х значення 6, це значення буде надане і змінній а. Оператор д:=х здійснює присвоєння значення 6 імені функції д. Точка повернення з функції д(12) є точкою входу до функції Т(6), в якій знову змінюється значення локальної змінної х.

Це значення стає рівним 7 і присвоюється імені функції т. Точкою повернення з функції і є оператор присвоєння значення 7 змінній Ь. Отже, в результаті роботи програми змінна а набуде значення 6, а змінна Ь — значення 7, Значення змінних та вміст стеку під час виконання програми можна переглядати у вікнах Маісіїез і Саіі Зіаск, що відкриваються за допомогою меню ОєЬид.

Таблиця 4,5. Стан оперативної пам'яті під час виконання програми ех4_5

Оператор, що **виконується**	Глобальна змінна **а**	Глобальна змінна Ь	Локальна змінна х	Стан вікна перегляду стеку
До початку виконання програми	IMпомі icienTITiez		11пкпомпЮепсіііег	
Веділ	0	0	Їїпкпомп icienTITier	ех4_5
а: =12:	12	0	Цпкпомі icieпїїпег	ех4_5
Ь:=Т(д(12))	12	0	12	9(12) ех4 5
х:=х СПУ 2; д:=х:	6	0	6	9(6) ех4 5
Х;=х+1; Т:=х;	6	0	7	Т(7) ех4 5
МгіТе1п('а=',а'Ь=',Ь): єпсі.	6	7	Ипкпомп icienШ'ier	ех4_5
Після завершення програми	6	7	IIпкпомп і СІЄТІСІ Ті ег	

4.1.7. Процедурні типи. Підпрограми як параметри

Основне призначення *процедурних типів* полягає в тому, щоб надати програмісту можливість передавати процедури або функції як фактичні параметри під час виклику інших процедур або функцій. Потребу у таких параметрах продемонструємо на прикладі задачі табулювання математичних функцій на заданому відрізку (приклад 4.6).

Приклад 4.6_

Припустимо, що треба надрукувати таблиці значень декількох функцій на відрізку *[а; Ь]* із кроком к. Нехай це буде многочлен $x^3 - x + 1$, степенева функція $e^{*\sim 3}$ та тригонометрична функція зіп *х*. Цю задачу можна було б ефективно розв'язати за допомогою процедури, що виводить значення довільної функції на заданому відрізку. Зрозуміло, що сама функція має бути аргументом цієї процедури, тобто процедура повинна мати параметр типу «функція». Оскільки параметри підпрограм оголошуються у вигляді пар <їм'я параметра>:ідентифікатор типу», то типу «функція» треба надати певний ідентифікатор, що можна зробити лише через оголошення цього типу в розділі Type. Існує чотири варіанти синтаксису оголошення процедурного типу, наведемо їх:

```
Type
«ідентифікатор типу> = ргосесііиге(<оголошення параметрів»);
```

```
«ідентифікатор типу> = ргосесіиге;
«ідентифікатор типу> = Типсііоп(«оголошення параметрів>);<тип>;
«ідентифікатор типу> = Типсііоп:<тип>;
```

Тут «ідентифікатор типу> — ім'я процедурного типу; ргосесіиге та ТипсТіоп — зарезервовані слова; «оголошення параметрів» - список параметрів (його синтаксис збігається із синтаксисом списків параметрів в оголошеннях підпрограм); <тип> - довільний неструктурований тип значення, що його повертає функція. Імена підпрограм в оголошеннях процедурного типу не зазначаються.

Повернімося до задачі табулювання математичних функцій на певному відрізку. Йдеться про функції однієї дійсної змінної. Всі такі функції мають один параметр типу геаі та повертають значення типу геаі, а отже, можуть розглядатись як функції одного процедурного типу. Оголосимо цей тип:

```
Type Типе = ТипсТіоп(х:геа1):геа1;
```

Підпрограми, що є параметрами процедур, слід компілювати з використанням *дальньої моделі пам'яті.* Модель пам'яті визначає спосіб виклику підпрограм. *Ближня модель* припускає виклик підпрограм лише в межах одного сегмента коду. Дальня модель пам'яті дозволяє звертатися до процедур і функцій з будь-якого сегмента і використовується не лише в оголошеннях підпрограм-аргументів, а й в оголошеннях підпрограм, що викликаються з інших модулів. Дальню модель можна встановити за допомогою ключового слова Таг, що вводиться після заголовка підпрограми. Наприклад:

```
Типсііоп «ім'я>(«список параметрів>):«тип>; Таг;
```

Встановити дальню модель можна також за допомогою директиви компілятора Гогсе Таг саііз, що записується в програмі у вигляді коментарів {$Р+} та {$Р-}. Коментар {$Р+} встановлює режим компіляції в дальній моделі, а коментар {$Р-} скасовує цей режим. Наприклад,

```
{$P+}
Типсііоп «ім'я>(«список параметрів»):«тип>;
Ьеділ
 «тіло функції»
єпсі:
{$P-}
```

Перед тим як навести код програми ех4_6, що здійснює табулювання функцій, зауважимо, що в мові РазсаІ не можна використовувати стандартні математичні функції як аргументи підпрограм. Якщо в цьому виникає потреба, слід написати власну «функцію-оболонку» з іншим ім'ям. Наприклад, у програмі ех4_6 для стандартної математичної функції зіп використовується функція-оболонка Зіпиз. Результати роботи програми табулювання функцій зображено на рис. 4.6.

```
ргодгат ех4_6;                          {табулювання функцій}
Type Типс=ТипсТіоп(х:геа1):геа1            {процедурний тип}
уаг 1омег.иррег.зТер:геаі;              {межі відрізка, крок}
```

```
I======================= ПОЛІНОМ ========================)
Типсііоп Ро1упот(х:геа1):геа1;  Таг:
Ьеділ
  Ро1упот:=здг(х)*х-х+1;
єпсі;
{=================== степенева функція ====================}
ТипсТіоп ЕхропепТ(х:геа1):геа1;  Таг;
Ьеділ
  ЕхропепТ;=Ехр(х-3);
єпсі;

ТипсТіоп 5іпиз(х:геаі):геа!;  Таг;
Ьеділ
  Зіпиз:=зіп(х);
єпсі;
{============= процедура табулювання функції Т ============}
ргосесіиге РгїпТ(а,Ь,іі:геа1 :Т:Типс;5:зТгіпд);
  {а - нижня межа відрізка,Ь - верхня межа. Н - відстань
   між точками. Т - ім'я функції, з - текстова назва функції}
уаг х:геа1;              {значення аргументу Т. параметр циклу}
Ьеділ
  мгіТе1п('==================');          {виведення заголовка}
  мгіТе!п('    х     |   [1],5):                          {таблиці}
  МП Те]П('==================');
  х:=а;
  міліє х<Ь сіо        {поки не досягнуто правої межі відрізка}
  Ьеділ
    мгіТе(х:6,'|[1]);
    мгіТе(Т(х):10:6);      {обчислити і вивести значення Т(х)}
    мгіТеІп:
    х:=х+Н:                      {збільшити значення аргументу}
  єпсі;
єпсі;
1================== основна програма ====================}
Ьеділ
  мгіТе!п('епТег Іомег.иррег Ьоипсіз апсі зТер');
  геасії п(1 омег, иррег.зТер);
  {табулювання многочлена}
  РппїОомег.иррег.зТер.РоІупот, 'х^3-х+Г );
  {табулювання експоненти}
  РгіпТ(1омег,иррег.зТер,ЕхропепТ.'е^(х-3)');
  {табулювання синусу}
  РгіпТ(1 омег,иррег.зТер,5іпиз,'зіп(х)');
єпсі.
```

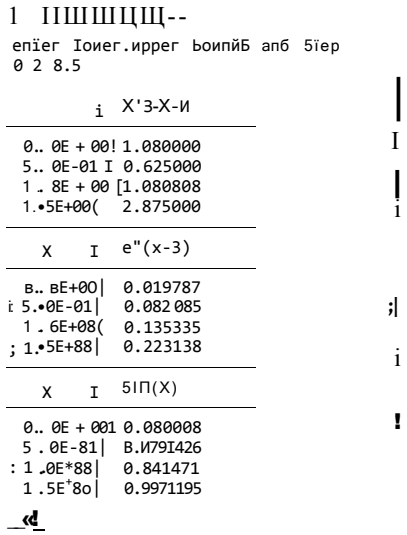


Рис. 4.6. Результати роботи програми ex4_6. Табулювання функцій

4.2. Рекурсія

Рекурсивні означення дозволяють за допомогою скінченного висловлювання означити нескінченну множину об'єктів і тому на таких означеннях ґрунтується потужний математичний апарат. Аналогічно, за допомогою скінченної рекурсивної програми можна, не використовуючи конструкцій повторення, описати нескінченні обчислення, і тому рекурсія є потужним інструментом програмування. Необхідним і достатнім засобом реалізації рекурсії є підпрограма. Розгляду рекурсивних підпрограм і присвячено даний розділ.

4.2.1. Рекурсивні означення та підпрограми

Означення того чи іншого поняття задає, як правило, певну множину об'єктів через виокремлення елементів інших множин. Наприклад, означення «студент - це людина, що навчається у вищому навчальному закладі» задає множину студентів, виокремлюючи деякі елементи множини людей. Означення називається *рекурсивним,* якщо воно задає елементи множини за допомогою інших елементів цієї самої множини. Наприклад, означення «натуральне число п зветься парним, якщо *n* = 2 або *n* - 2 — парне число» є рекурсивним означенням множини парних натуральних чисел. Об'єкти, що задані рекурсивним означенням, також називаються *рекурсивними.* Нарешті, *рекурсією* називається реалізація рекурсивних означень за допомогою підпрограм. Інакше кажучи, рекурсія - це такий спосіб організації обчислювального процесу, за якого процедура або функція зверта-

ється сама до себе. Такі звернення називаються *рекурсивними викликами,* а підпрограма, що містить рекурсивні виклики, — *рекурсивною.*

Приклад 4.7

Одним із найпростіших прикладів рекурсії може стати функція обчислення факторіала. Нагадаємо, що факторіалом *n*\ натурального числа *n* називається добуток усіх цілих чисел від одиниці до *n.* Вважають також, що 0!=1. У прикладі 3.7 було реалізовано нерекурсивний спосіб обчислення факторіала за допомогою циклу:

```
Tacioriai:=1;
  тог i:=2 io n cio
    Tacioriai:=Tacioria1*i;
```

Наведений код реалізує такий принцип обчислень. Покладемо спочатку 1!=1, потім 1! помножимо на 2 і отримаємо 2!, потім 2! помножимо на 3 і отримаємо 3! тощо. Отже, обчислення факторіала зводиться до багаторазового застосування рекурентної формули *n*! = *(n* -1)! • *n* Ця формула є рекурсивною, оскільки означує «факторіал через факторіал», проте спосіб її застосування може бути як рекурсивним, так і нерекурсивним. Рекурсивне обчислення п\ є не висхідним: «обчислимо спочатку 1!, потім - 2! і т. д.», а низхідним: «обчислимо (п - 1)! і отримане значення помножимо на *n».* При обчисленні (я - 1)1 буде застосоване те саме правило: «обчислимо (п - 2)! і отримане значення помножимо на п- і»; це саме правило буде застосоване при обчисленні (п - 2)! і т. д. Для того щоб це правило не застосовувалося нескінченно, необхідно визначити умову зупинення, що називається *умовою завершення рекурсії.* Такою умовою буде рівність *n* нулеві. Дамо повне рекурсивне означення факторіала:

$$I \quad \begin{cases} \Gamma(n-1)!\text{-и}, & \text{якщо га} > 0; \\ \backslash i, & \text{якщо я} = 0. \end{cases}$$

Реалізуємо щойно наведене означення рекурсивною процедурою у програмі Єх4_7.

```
Рrодгат ех4_7;                    {обчислення факторіала числа}
уаг пііпТедег;    {число, факторіал якого необхідно обчислити}
Типсііоп Р(п:inТедег):inТедег;              {оголошення функції}
Ьеді п
   iP п=0 iНеп Р:=1
          еізе Р:=Р(п-1)*п;       {рекурсивний виклик функції}
єпсі:
Ьеді п                                        {основна програма}
  мгііе!п(1Епіег п 1):
  геасії п Сп);
  мгііе!п(•п!=',Р(п)):
єпсі.
```

Роботу рекурсивної функції розглянемо на прикладі обчислення 4!. Основна програма звертається до функції Р(4). Виклик функції Р(3) відбудеться ще до першого виконання операції множення, після цього, теж ще до операції множення,

з функції PC3) буде викликана функція P(2) тощо. Процес рекурсивних викликів припиниться у функції P(0), що поверне одиницю. Ця одиниця стане значенням лівого операнда операції P(0)*1 у функції P(1). Результат операції буде повернутий функцією P(1) у функцію P(2) тощо. Отже, під час рекурсивних викликів функцій P виконання операцій множення відкладатиметься, оскільки один із множників ще не відомий. Добутки будуть обчислені при поверненні із функцій.

На рис. 4.7 проілюстровано процес рекурсивного обчислення 4!, а у табл. 4.6 зображено стан стеку та значення змінної п під час обчислення 2!.

РИС. 4.7. Рекурсивне обчислення факторіала

Таблиця 4.6. Стан оперативної пам'яті під час рекурсивного обчислення факторіала

Оператор, що виконується	Значення змінної п	Стан стеку
игПе1пСп! = \Г(п));	2	ex4 7
=Р(п-1)*п;	2	P(2), ex4_7
=Р(п-1)*п;	1	P(1), P(2), ex4 7
=1	0	P(0), P(1), P(2), ex4_7
Enci	0	P(0), P(1), P(2), ex4 7
Enci	1	P(1), P(2), ex4_7
Enci	2	P(2), ex4 7
Enci.	2	ex4 7

Отже, в рекурсивних підпрограмах можна виділити два процеси: *рекурсивне занурення* підпрограми у себе, що відбувається доти, доки параметр не сягне граничного значення, *та рекурсивне повернення* з підпрограми, що відбувається доти, доки параметр не сягне початкового значення. Після виклику підпрограми з аргументом *n* виконується ще *n* - 1 викликів, і загальна кількість незавершених викликів сягає *n*. Величина, що характеризує максимальну кількість незавершених рекурсивних викликів, називається *глибиною рекурсії*.

Від глибини рекурсії значною мірою залежить час виконання програми та об'єм потрібної стекової пам'яті. Значні витрати стекової пам'яті пов'язані із тим,

що в рекурсивній підпрограмі, як правило, оголошується численна кількість локальних об'єктів: змінних, констант, типів, вкладених підпрограм тощо. Кожного разу, коли підпрограма викликається рекурсивно, виділяється стекова пам'ять для всіх її локальних об'єктів, і тому велика глибина рекурсії може призвести до нестачі стекової пам'яті.

Якщо в тілі підпрограми здійснюється більше ніж один рекурсивний виклик, то надвеликими можуть стати не лише витрати стекової пам'яті, але і витрати часу. У цьому разі, можливо, перевагу варто надати нерекурсивному розв'язанню задачі. Неефективне застосування рекурсії продемонстроване у прикладі 4.8.

Приклад 4.8_

Розглянемо рекурсивну функцію, що обчислює *n*-й член послідовності чисел Фібоначчі. Ці числа визначаються рекурентним співвідношенням $/_{„} = /_{„\_|} + /_{п\_2}$ для $П > 0, /_1 = 1, /о = 0$.

```
Типсїіоп  РіЬСп:іпїедег):іпіедег:
Ьеділ
  іі п = 0 ТЇеп РіЬ:= 0
       еізе іТ п = 1 їНеп РіЬ:= 1
                     еізе  РіЬ:=РіЬ(п-1)+РіЬ(п-2);
епсі;
```

Кожен виклик функції РіЬ при $n > і$ породжує два вкладених виклики. У результаті відбуваються повторні обчислення тих самих величин і загальна кількість вкладених викликів зростає експоненціально. Цей процес ілюструє рис. 4.8, на якому числа відповідають значенням параметра п.

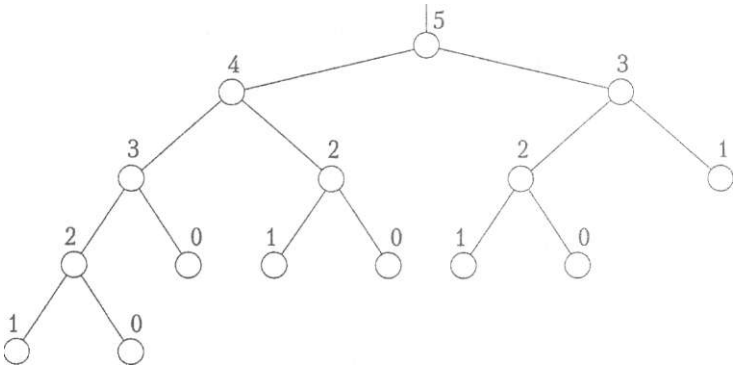


Рис. 4.8. Вкладені рекурсивні виклики функції, що повертає число Фібоначчі

Наприклад, виконання функції РіЬ(5) призведе до того, що виклик РіЬ(3) буде здійснено двічі, виклики РіЬ(2) та РіЬ(О) - тричі, а виклик РіЬ(1) - п'ять разів. Загальна кількість вкладених викликів сягне п'ятнадцяти. Тому для практичного використання така програма непридатна.

4.2.2. Приклади рекурсивних програм

У попередньому розділі було показано, що для деяких рекурсивних означень ефективний спосіб їх реалізації буде не рекурсивним. Але для інших задач рекурсія є найбільш природним та найбільш ефективним способом розв'язання. Розглянемо два класичні приклади таких задач: гру «Ханойські вежі» (приклад 4.9) та задачу швидкого піднесення до степеня (приклад 4.10).

Приклад 4.9

Є три стрижні з номерами 1, 2, 3. На стрижні 1 розміщена вежа з я дисків різного діаметра; нижній диск має найбільший діаметр, а діаметр кожного наступного диска менший від діаметра попереднього. Необхідно перенести всі диски на стрижень 3 так, щоб порядок їх розташування не змінився. Під час перенесення дисків слід дотримуватися таких правил: на кожному кроці переноситься лише один диск; більший диск не можна класти на менший.

Назва гри походить від давньої легенди, яка стверджує, що понад 1000 років тому в одному монастирі поблизу в'єтнамського міста Ханой ченці почали перекладати 64 золотих диски з одного стрижня на інший; коли вони закінчать цю роботу, настане кінець світу. Доведено, що мінімальна кількість переміщень дорівнює 2" - 1, де я — число дисків. Значення 2^{64} наближено дорівнює 10^{21}. Тому, якщо припустити, що кожної секунди ченці переносять один диск, то для перенесення всієї вежі потрібно більше 10^{i4} років!

Якщо вежа містить один диск, то розв'язання є очевидним і треба лише перенести цей диск на третій стрижень. Якщо на першому стрижні є два диски, то менший диск перенесемо з першого стрижня на проміжний, потім більший диск перенесемо на третій стрижень і, нарешті, менший диск з проміжного стрижня перенесемо на третій. За три дії обидва диски перекладено. Тепер припустимо, що роль меншого диска відіграє вежа з *n* - 1 менших дисків, а більший диск має номер я. Алгоритм гри залишається абсолютно незмінним, якщо замість словосполучення «менший диск» вживати фразу «вежа з я - 1 менших дисків».

Алгоритм гри «Ханойські вежі»

1. Перенести вежу з я - 1 дисків з першого стрижня на проміжний.
2. Перенести я-й диск з першого стрижня на третій.
3. Перенести вежу з я - 1 дисків з проміжного стрижня на третій.

Таким чином, задачу перенесення я дисків було зведено до задачі перенесення я - 1 дисків. Запрограмуємо щойно наведений алгоритм. З цією метою розробимо рекурсивну процедуру **м**оує (п, Тгот. іепір, оп), що реалізує перенесення *n* дисків із стрижня *гот* (вхідний стрижень) на *оп* (цільовий стрижень) з використанням стрижня *іетр* як проміжного. Зазначимо, що ролі стрижнів під час гри змінюються. Якщо спочатку проміжним є другий стрижень, то при виконанні кроку 1 алгоритму проміжним стає третій, а при виконанні кроку 3 — другий стрижень. Решту коментарів наведено в тексті програми ex4 8, результат роботи якої зображено на рис. 4.9.

```
ргодгат ех4_8;                                  {Ханойські вежі}
уаг п;іпТедег;                                 {кількість дисків}
 ================== ініціалізація даних ====================}
ргосесіиге  Іпіі;
Ьедіп
  мгіТе!п('Напоу Томегз');
  мгіТе!п('епТег пишЬег оТ сіізк ');
  геасііп(п);
  «гіТе1п('Ігапзрогі ргосезз:');
епсі;
 =================== перенесення ДИСКІВ ====================}
ргосесіиге   МОУЄ(І :іпіедег;Тгош,їетр,оп:сНаг):
•І - КІЛЬКІСТЬ дисків;Тгот,Тешр,оп - назви стрижнів}
Ьедіп
  іТ і>0 Тііеп        {перенести один диск та вийти із процедури}
Ьедіп
        {перенести і-1 дисків із вхідного стрижня на проміжний}
    МОУЄ(І -1,Тгош,оп.Тешр);
    {перенести найбільший диск із вхідного стрижня на цільовий}
    мгіТе1п('ШОУЄ сіізк '.і.' Тгош '.Тгош,' То '.оп);
       {перенести і-1 дисків із проміжного стрижня на цільовий}
    ИОУЄ(І-1,Тешр,Тгош,оп);
  епсі;
епсі;
 =================== основна програма =====================}
Ьедіп
  ІпіТ;
  ИОУЄ(П,'1','2'.'3');        {виклик рекурсивної процедури}
епсі.
```

```
                                    /       -піхі
Напоу Іомег5
епіег питЬег оГ Йіїк
3
ііrапзрогі ргосе55:
тоне йіік 1 Ггоп 1 Іо 3
тоне ЙІ5И 2 Ггого 1 їо 2
тоие аj5к 1 Ггот 3 іо 2
тоне Йі5к 3 Ггот 1 Іо 3
точе Йі5к 1 Ггот 2 Го 1
тоїіе ()І5к 2 Ггот 2 іо 3
тоие Йіїк 1 Ггот 1 Іо 3
```

РИС. 4.9. Результати роботи програми ех4_ Ханойські вежі

Приклад 4.10_

«Індійський алгоритм» піднесення числа *x* до натурального степеня *n* реалізує гаке рекурсивне означення степеня числа:

$$x'' = \begin{cases} \backslash X & \Pi = 0; \\ \ddot{\imath}x, & n = 1; \end{cases}$$

Тривіальний спосіб піднесення числа x до степеня *n* потребує, щоб виконувались *n*-1 операцій множення: *x* множиться на *x,* добуток знову множиться на *x,* і так я - 1 разів. Застосування індійського алгоритму дає можливість значно скоротити кількість операцій. Наприклад, обчислення x^{10} за індійським алгоритмом потребує лише чотирьох операцій множення замість дев'яти. Справді, $x^{10} = (x^5)^2 = (x(x^2)^2)^2$. Тобто, при обчисленні x^{10} операції виконуватимуться в такому порядку:

1) обчислюємо x^2;
2) x^2 підносимо до квадрата і отримуємо *x ;*
3) x^4 множимо на x і отримуємо x^5;
4) x^5 підносимо до квадрата і отримуємо x^{10}.

Наведене вище рекурсивне означення степеня числа може бути реалізоване рекурсивною процедурою. При рекурсивному зануренні така процедура обчислюватиме значення *n* спу 2, а при виході з рекурсії обчислюватиметься множник уптоа 21 виксшуватиметься операція множення. Оскільки *n* тосі 2 дорівнює нулю, якщо *n* — парне, і дорівнює одиниці, якщо *n* — непарне, то величина $x^{\text{n тос1 2}}$ дорівнюватиме 1 за парних значень *n* і дорівнюватиме *x* за непарних значень *n.* Отже, наведемо деталізований індійський алгоритм піднесення до степеня та його програмну реалізацію. Результати роботи програми зображено на рис. 4.10.

Алгоритм обчислення *x"*

1. Якщо показник степеня дорівнює нулю, то значення x" покласти рівним одиниці та вийти з рекурсії, інакше виконати дії, зазначені в пунктах 2-5.
2. Якщо показник степеня дорівнює одиниці, то значення x* покласти рівним x та вийти з рекурсії, інакше виконати дії, зазначені в пунктах 3-5.
3. Обчислити значення $x^{*\ 2}$ та піднести його до квадрата.
4. Якщо показник степеня непарний, то обчислене в пункті 3 значення помножити на число x, а отриманий результат вважати значенням x".
5. Інакше, коли показник степеня парний, то обчислене в пункті 3 значення вважати значенням x*.

```
ргодгат ех4_9;    {індійський алгоритм піднесення до степеня}
уаг Ьазе.ехропепі:ІпТедег;          {число, показник степеня}
ргосесіиге Іпіі;
Ьеділ
мгііе!п(' Іпсііап аІдогіШГ);
  мгііє1п('епіег Ьазе, ехропепі');
  геасії п(Ьазе.ехропепі);
епсі;
І================ піднесення до степеня ====================}
типсііоп ром(х.п:іпіедег): 1 опділ;
уаг і: іпіедег;                              {допоміжна змінна}
Ьеділ
  ІТ п=0 ТНеп ром:=1
         еізе
  іТ п=1 іЬеп ром:=х                  {рекурсивне повернення}
         еізе
```

```
        Ьеділ
          і;=5дг(ром(х. п сііу 2));     {рекурсивне занурення}
          іТ оскКп) іНеп ром;=і*х      {рекурсивне повернення}
                    еізе ром;=і;       {рекурсивне повернення}
        єПСі;
єпсі;
;================ основна програма ====================}
Ьеділ
  Іпіі;
  мгііе1п(Ьа$е,.ехропепі,'=',рои(Ьа$е,ехропепі));
епсі.
```

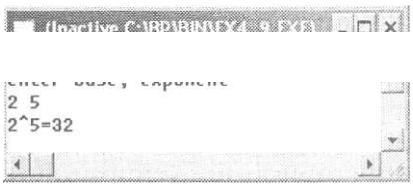


Рис. 4.10. Результати роботи програми ех4_9. Швидке піднесення до степеня

Тепер спробуємо описати залежність глибини рекурсії від значення степеня я. У кожному наступному вкладеному виклику значення аргументу я менше від попереднього значення принаймні вдвічі. Оскільки за умови я = 1 розпочинається рекурсивне повернення, то таких зменшень значення аргументу я не може бути більше $1o§_2$ я. Отже, глибина рекурсії не перевищує 1o§ 2 я- Таку глибину можна вважати ознакою високої ефективності алгоритму. Під час виконання кожного рекурсивного виклику відбувається не більше ніж одна операція ділення, піднесення до квадрата та множення, тому загальна кількість арифметичних операцій не перевищує $ЗIo§_2$ я Якщо величина я достатньо велика, це суттєво менше, ніж *n* - 1 множень. Наприклад, за умови я = 1000 кількість арифметичних операцій приблизно дорівнює ЗО.

Зауважимо, що для деяких значень я обчислити *x"* можна швидше, ніж це робить індійський алгоритм. Наприклад, для знаходження $х^{15}$ за цим алгоритмом потрібні шість операцій множення, хоча можна за допомогою трьох операцій обчислити $де^5$ і отримане значення піднести до куба за допомогою ще двох операцій (разом — п'ять множень). Проте створення ефективного алгоритму, що обчислює довільний степінь за допомогою мінімальної кількості операцій множення, є складною і досі не розв'язаною задачею.

4.2.3. Випереджальне оголошення процедур і функцій

Вище було розглянуто лише *пряму рекурсію,* за якої підпрограма містить виклики самої себе. У програмуванні використовується також *непряма рекурсія,* коли підпрограма містить виклики інших підпрограм, що, у свою чергу, містять виклики даної підпрограми. Приклади непрямої рекурсії будуть наведені в цьому розділі.

Приклад 4,11

Розглянемо рекурсивний спосіб обчислення значень функцій синуса та косинуса. Використаємо такі тотожності:

$$\text{sin}x = 2\text{sin}(x/2)\,\text{cos}(x''2), \qquad \text{cos}x = \text{cos}^2\,(x/2)\text{-sin}^2\,(x/2).$$

Згідно з цими тотожностями функції синус та косинус викликають один одного і тому є *взаємно рекурсивними.* За правилами мови РазсаІ, кожний ідентифікатор має бути оголошений до його використання. Оскільки функція синус викликає функцію косинус., то функція косинус має бути оголошеною до оголошення функції синус. Але і функція косинус, у свою чергу, викликає функцію синус, і тому функція синус має бути оголошеною до оголошення функції косинус. Розв'язати цю суперечність дає можливість *випереджальне оголошення* процедур і функцій, що реалізується конструкцією Тогмагсі.

Якщо у програмі є підпрограми, що викликають одна одну, то спочатку в розділі оголошень програми або підпрограми вищого рівня записуються лише заголовки кількох із них (якщо таких підпрограм дві — то однієї з них), а замість їх тіла пишеться директива Тогшгсі, тобто «попереду»:

```
ргосесііиге <ім'я>(<оголошення параметрів>); Тогу/агсі;
Типсііоп <ім'я>(<оголошення параметрів») :<тип> Тогмагсі;
```

Така підпрограма стає відомою іншим підпрограмам навіть без фактичної реалізації її тіла. Решту підпрограм розташовують так, щоб вони містили виклики лише тих підпрограм, заголовки яких (разом із тілом чи директивою Тогмагсі) було записано вище. Підпрограми, що їх заголовки записано без тіла, мають бути реалізовані нижче. Така реалізація розпочинається зі скороченого заголовка:

```
ргосесііиге <ім'я>;
Типсііоп <ім'я>:
```

Оголошення параметрів і типу значення, що його повертає функція, у скороченому заголовку відсутні.

Наведемо програмну реалізацію рекурсивного обчислення синуса та косинуса. Рекурсивні функції оголосимо як sіпе(х) і созіп(х). Умовою виходу з рекурсії вважатимемо рівність аргументу х нулю з точністю до величини ерз. Отже, при поверненні з найглибшого рекурсивного виклику необхідно буде нерекурсивно обчислити синус та косинус малого числа. Використаємо для цього перші два члени розвинення тригонометричних функцій у степеневий ряд. Результати роботи програми зображено на рис. 4.11.

```
ргодгат ех4_10:
уаг х.ерз:геаі;

Типсііоп созіп(х:геа1):геа1: іогмаггі
{ — = = = = = = = = = = = = — : >
Типсііоп зіпе(х:геа!):геа1;
уаг з:геаі;
```

```
седіп
  і і  (аЬз(х)<ерз)
  їЬеп  з:=х*(1-х*х/6)
  еізе  з:=2*зіпе(х/2)*созіп(х/2);
  зіпе:=з;
епсі;

"ЛІСІІОП  созіп;
.аг с : гєаі;
зеді п
  іТ  (аЬз(х)<ерз)
    іНеп  с:=1-х*х/2
    еізе с: = здг(со5Іп(х/2))~здг(5іпе(х/2));
    созіп:=с;
  епсі;

зеді п
   мгіїе1п('іпсіігесі  гесигзіоп[1]);
   мгіїе('епіег х,  ерз  ');
   геасПп(х.ерз);
   ипІеІпС'зіп(х)=' ,зіпе(х) :6:6);
   мг1ie1п('соз(х)=',созіп(х):6:6);
епсі.
```

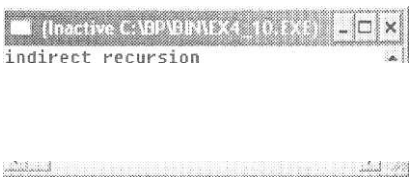


РИС. 4.11. Результати роботи програми ех4 Ю. Обчислення тригонометричних функцій за допомогою непрямої рекурсм

Висновки

- Один з ефективних способів створення великих програм, технологія низхідного проектування, полягає в їх конструюванні за принципом «розділяй і панує»: програма розглядається як набір маленьких фрагментів, кожний з яких виконує певну логічно завершену дію, може бути виконаний декілька разів і є більш керованим, ніж програма у цілому. Такий невеликий фрагмент програмного коду називається підпрограмою.
- Підпрограма позначається ідентифікатором. Використання в тексті програми ідентифікатора підпрограми називається викликом підпрограми і приводить до виконання всіх операторів, що входять до її складу.
- Мови, в яких реалізовано механізми використання підпрограм, називаються процедурно-орієнтованими.

- У мові РаscaI існує два різновиди підпрограм — процедури та функції. Функція відрізняється від процедури тим, що вона повертає деяке значення в точку її виклику. Іншими словами, під час виклику функції її ім'я може інтерпретуватись як ім'я деякої змінної величини і використовуватись у виразах.

4- Вбудовані або стандартні процедури та функції є частиною мови, вони можуть викликатися без попереднього оголошення у програмі.

- Параметрами називаються змінні, за допомогою яких відбувається передача даних до підпрограми, що викликається, із програмного блоку, що здійснює виклик.

4 Оголошення підпрограми складається із заголовка та тіла. Заголовок процедури містить її ім'я та, можливо, список параметрів. У заголовку функції, крім її імені та списку параметрів, повинен вказуватися тип значення, що його повертає функція. Тіло підпрограми містить оператори, які будуть виконані під час її виклику.

4 Параметри, що їх імена вказані в заголовку підпрограми, називаються формальними. За допомогою формальних параметрів реалізується механізм, який дає можливість виконувати підпрограму з різними вхідними даними.

4 Значення, що замінюють формальні параметри під час виклику підпрограм, називаються фактичними параметрами або аргументами. Під час виклику підпрограми між її фактичними та формальними параметрами встановлюється взаємно однозначна відповідність за кількістю параметрів, їх типами та порядком використання.

4- У підпрограму можна передати значення будь-якого скалярного типу, структурованого типу, а також типу покажчика.

4 Функція повертає в точку виклику лише одне значення неструктурованого типу.

4 Тіло функції має містити оператор присвоєння її імені значення, яке ця функція повертає.

4 Кожний ідентифікатор у програмі характеризується областю дії імені, або областю видимості. Область видимості ідентифікатора - це область програми, в якій на даний ідентифікатор можна посилатися. Область дії іменування поширюється від тієї точки, де ідентифікатор було оголошено, до кінця того програмного блоку, в якому це оголошення відбулося.

4 Ідентифікатори, оголошені всередині процедури або функції, називаються локальними. Зокрема, локальними є будь-які параметри підпрограми.

4 Ідентифікатор називається глобальним по відношенню до даної підпрограми, якщо його оголошено в основній програмі або в підпрограмі вищого рівня, яка містить у собі дану підпрограму.

4 Формальні параметри поділяються на параметри-значення, параметри-змінні, параметри-константи та нетипізовані параметри-змінні.

4 У разі, коли параметр оголошено як параметр-значення, під час виклику підпрограми обчислюється значення відповідного аргументу і копія отриманого

результату передається підпрограмі. Зміна параметрів-значень усередині підпрограми не впливає на значення змінних, що вказуються в операторі виклику підпрограми як її аргументи.

- Якщо параметр оголошено як параметр-змінну, то до підпрограми передається покажчик на цей параметр, тобто адреса відповідної змінної в сегменті даних оперативної пам'яті. Модифікація параметра-змінної приведе до модифікації фактичного параметра. У заголовку підпрограми перед іменем параметра-змінної потрібно записати зарезервоване слово уаг.

4 Для параметра-константи копія значення відповідного аргументу під час звернення до підпрограми не створюється. Значення такого параметра не можна змінювати в тілі підпрограми. У заголовку підпрограми перед іменем параметра-константи потрібно записати зарезервоване слово сопзі.

- Якщо формальний параметр є нетипізованим параметром-змінною, то відповідний йому фактичний параметр може бути покажчиком на змінну довільного типу. У заголовку підпрограми перед іменем параметра-змінної слід записати слово уаг, але не потрібно вказувати тип параметра.

4- Перша команда підпрограми називається точкою входу, а оператор, що продовжує виконання програми по завершенні роботи підпрограми, називається точкою повернення з підпрограми.

4 Програмний стек — це ділянка пам'яті, що використовується для тимчасового збереження параметрів, локальних змінних та адрес точок повернення з підпрограм.

4 Параметрами підпрограм можуть бути не тільки змінні, але й інші підпрограми. Параметр-підпрограма є змінною процедурного типу. Множиною значень процедурного типу є множина підпрограм, що мають спільний список формальних параметрів та, якщо йдеться про функції, повертають значення того самого типу.

4 Означення називається рекурсивним, якщо воно задає елементи множини за допомогою інших елементів цієї ж множини. Об'єкти, задані рекурсивним означенням, також називаються рекурсивними.

4 Рекурсія — це такий спосіб організації обчислювального процесу, за якого процедура або функція звертається сама до себе. Такі звернення називаються рекурсивними викликами, а підпрограма, що містить рекурсивні виклики, - рекурсивною.

4 У рекурсивних підпрограмах можна виділити два процеси: рекурсивне занурення підпрограми в себе, що відбувається доти, доки її параметр не сягне граничного значення, та рекурсивне повернення з підпрограми, що відбувається доти, доки її параметр не сягне початкового значення.

4 Максимальна кількість незавершених рекурсивних викликів під час виконання рекурсивної підпрограми називається глибиною рекурсії.

- Використання рекурсії може призвести до нестачі стекової пам'яті та уповільнення швидкості виконання програми. Тому не варто застосовувати рекурсію тоді, коли задача має очевидне ітеративне розв'язання.
- У разі непрямої рекурсії підпрограма містить виклики інших підпрограм, що, у свою чергу, містять виклики даної підпрограми. Якщо декілька підпрограм містять непрямі рекурсивні виклики, то до деяких з них має бути застосоване випереджальне оголошення.

Контрольні запитання та завдання

1. У чому полягає технологія низхідного проектування програм?
2. Що таке підпрограма та які переваги дає використання підпрограм?
3. Дайте визначення процедури та функції.
4. Наведіть синтаксис оголошення процедури та функції.
5. Чим тіло процедури відрізняється від тіла функції?
6. У чому полягає відмінність виклику процедури від виклику функції?
7. Що таке параметри підпрограми?
8. У чому полягає відмінність формальних параметрів від фактичних?
9. Які різновиди формальних параметрів існують у мові РазсаІ?
10. Як повертаються значення з функції? Скільки значень можна повернути?
11. Які процеси відбуваються в оперативній пам'яті під час виклику підпрограми?
12. Що таке точка входу до підпрограми та точка виходу з неї?
13. Яке призначення має програмний стек? Який його обсяг? Як цей обсяг можна змінити?
14. Яке призначення мають процедурні типи? Як їх використовувати?
15. Що таке рекурсивне визначення і рекурсивний об'єкт?
16. Визначити поняття рекурсії. Що таке рекурсивна підпрограма?
17. Як визначається глибина рекурсії?
18. Як обмежити послідовність вкладених викликів рекурсивної підпрограми?
19. Коли рекурсія неефективна і коли її необхідно уникати?

Вправи

1. У наведених нижче твердженнях зробити доповнення.
   1.1. На відміну від процедури, функція_____.
   1.2. Змінна, що є видимою лише в тілі тієї підпрограми, в якій її оголошено, називається_____.
   1.3. Змінна, яку оголошено поза межами будь-якої підпрограми, називається

1.4. Підпрограма, що викликає сама себе, називається

1.5. Частина програми, в якій на змінну можна посилатися, називається

2. Як ви гадаєте, процедури reacïï п і мгіїеі п мають параметри-значення або параметри-змінні?
3. Написати функцію обчислення меншого з двох значень.
4. Написати функцію визначення парності цілого числа.
5. Написати процедуру обміну значеннями між двома змінними.

Задачі

1. Обчислити значення функції /(х), використавши розвинення функції зіпд: у ряд Тейлора. Аргумент *х* змінюється від -2 до 2 з кроком 0,5. Визначити похибку.

$$/00 = \begin{cases} |\text{зіп}^2 x - 5|\text{п}x, & 0 < x < 1; \\ \text{зіп}^3 x + \text{ЗП2А-}, & -2 < Л: < 0. \end{cases}$$

$$\text{зіпх} = x - x^3/3! + x^5/5! - \ldots.$$

2. Задано дійсне число є > 0. Обчислити $\int_a^b \sqrt{4 + x^2}\,dx$ з точністю до є > 0 за формулою трапецій.
3. Методом ділення відрізка навпіл з точністю до є > 0 знайти корені рівняння */(х)* = *х* + 1п(х + 0,1) - 0,5 = 0 на інтервалі [0; 2].
4. Знайти корінь рівняння на відрізку [0; *п/2]* із заданою користувачем точністю. Застосувати декілька ітераційних методів (ділення відрізка навпіл, хорд тощо).

$$2\,\text{зіп}^2 x - \frac{3\cos^2 x}{4} = n$$

5. Задано довжини чотирьох відрізків. Визначити кількість трикутників, що їх можна утворити з таких відрізків.
6. Прочитати координати точок *А, В, С, Б* і визначити, чи є замкнена ламана *АВСБА:*

   а) чотирикутником;

   б) опуклим чотирикутником.
7. Реалізувати рекурсивний алгоритм множення двох натуральних чисел, використовуючи рекурентне співвідношення

$$a \cdot b = \begin{cases} b, & a = 1, \\ (a-1) \cdot b + b, & a > 1. \end{cases}$$

Для числа, що введене із клавіатури, визначити суму цифр за допомогою рекурсивної функції.

Обчислити функцію Аккермана за її рекурсивним означенням:

$$A(n, ni) = \begin{cases} m+1, & n = 0, \\ A(П\text{-}1,1), & ПФ\ 0, \quad 772 — 0, \\ A(n\text{-}1)_{;}\ A(и,т\text{-}1)), & и>0, \quad m>0. \end{cases}$$

10. Один із варіантів алгоритму Евкліда для обчислення найбільшого спільного дільника чисел *а* і *Ь* ґрунтується на обчисленні рекурентної послідовності {р,}, д е = $max\{a,Ь\}, p_2$ = тіп{а,6}, p_n-p_{n_2} той $p_{n\_}$\ при $n>2$. Шуканим є останнє ненульове значення послідовності. Реалізувати цей алгоритм у вигляді рекурсивної функції.
11. Реалізувати генератор послідовності псевдовипадкових чисел {V*} на основі рекурентного співвідношення V;- = + с) тосі *m,* де *а, Ь,с,т* — довільні натуральні параметри. Перші два значення, У\ і $У_2$, задаються випадково. Підібрати значення параметрів, за яких послідовність є схожою на випадкову.
12. Написати програму знаходження кореня рівняння $f(x) = 0$ на відрізку [-1; 1], використовуючи функцію як параметр.
13. Вивести всі пари дружніх чисел, що не перевищують заданого натурального числа. Два числа називаються дружніми, якщо кожне з них дорівнює сумі всіх дільників іншого, за винятком його самого.
14. Знайти всі тризначні члени послідовності, визначеної рекурентним співвідношенням -Ро = 5, $Pi = 1$, $P_2 = 1$, $P_{n+з} = P_{n+2} + P_{n+X} - P_n$, якщо $n > 0$.
15. Обчислити кількість сполучень за рекурентною формулою біноміальних коефіцієнтів:

$$C^{к}_{,,} = \begin{cases} '0, & к > n, \\ 1, & к = 0 \text{ або } к = n, \\ c^{к}_{,,}\text{-}i+c^{к}_{п}\text{:}1 & о <к <, \end{cases}$$

16. У країні Тутті-Фрутті діє фінансова піраміда. Кожен з представників племені Тутті залучає до участі в піраміді трьох представників племені Тутті і чотирьох представників племені Фрутті, а кожен з представників племені Фрутті - п'ятьох представників племені Тутті і двох представників племені Фрутті. Кожен з представників племені Тутті перераховує «нагору» 5 тугриків і ЗО % від надходжень, отриманих «знизу», а кожен з представників племені Фрутті - 10 тугриків і 20 % від надходжень, отриманих «знизу». Визначити прибуток засновника &-рівневої піраміди, якщо він є представником племені Тутті.

Розділ 5

Програмування графіки

- **Графічний режим, його ініціалізація**
- **Графічна система координат**
- **Графічні процедури й функції**
- **Використання кольорів і стилів**
- **Зображення текстової інформації у графічному** режимі
- **Побудова фрактальних зображень**

5.1. Ініціалізація графічного режиму

Відеоадаптер персонального комп'ютера може працювати в одному із двох режимів — текстовому або графічному. У текстовому режимі на екрані дисплея відображаються лише символи. У графічному режимі мінімальним елементом зображення на екрані дисплея є *піксел,* або *графічна точка.* Зауважимо, що \УіпсІо\У5-програми працюють лише у графічному режимі, а використання текстового режиму є характерним для БОЗ-програм. Коли програму, що працює у текстовому режимі, запускають із середовища "^Уіпсіо'у/з, режим відеоадаптера може залишатися графічним, а текстовий режим буде земульований у межах вікна програми. Такий режим виконання БОЗ-програми називається віконним. Повноекранний режим виконання БОЗ-програми полягає у встановленні потрібного відеорежиму під час її запуску і у поверненні до початкового відеорежиму після завершення роботи. БОЗ-програма, що працює з графікою, може бути виконана лише у повноекранному режимі. Надалі у цьому розділі ми розглядатимемо лише графічні ВОЗ-програми.

Програма, що працює у графічному режимі, використовує *графічні драйвери* — файли, що містять інформацію про властивості відеоадаптерів. У середовищі Вогіапсі РазсаІ 7.0 графічні драйвери зберігаються у файлах, які мають розширення Ьді (Вогіапсі Сгаріїіісз Іпіегіасе).

Для різних типів відеоадаптерів використовуються різні графічні, драйвери, Визначальними характеристиками відеоадаптера є роздільна здатність, що визначається кількістю пікселів на екрані у горизонтальному та вертикальному вимірі, і кількість кольорів, якими може бути відображений будь-який піксел. Усі сучасні дисплейні адаптери належать до класу 5УСА (Зирег Уісіео СгарЬісз Аггау). вони мають граничну роздільну здатність понад 640х480 пікселів та дозволяють використовувати не менш ніж 256 кольорів. Для роботи із ЗУСА-адаптерами

придатні драйвери **5уда25бЬді** і **едауда.Ьді**. Драйвер **едауда.Ьді** не підтримує відеорежими із роздільною здатністю, що перевищує 640x480 пікселів, проте його використання гарантує сумісність програми майже із будь-яким графічним адаптером. Під час створення програм даного розділу припускатимемо, що використовується саме драйвер **едауда.Ьді**.

Крім графічних драйверів, під час роботи з графікою в середовищі Вогіапсі РазсаІ 7.0 використовується стандартний бібліотечний модуль СгарЬ. Він є бібліотекою підпрограм, що містить біля 80 графічних процедур і функцій, а також десятки стандартних констант і оголошень типів даних. Модуль йгарИ підключається до програми за допомогою оператора изез:

```
изез бгарН;
```

Модуль Сгарії міститься у файлі **...\ипіі5\дгарН**.іри, і щоб забезпечити можливість роботи із графікою, цей файл потрібно зробити досяжним для компілятора. Для цього шлях до файла модуля Огарії слід записати в полі Іпіі Oirecioriez вікна, яке відкривається за допомогою команди Opiiопз • Oirecїoriez.

Перед тим як виводити певні зображення, слід ініціалізувати графічний режим. Ініціалізація графічного режиму виконується процедурою, яка завантажує до оперативної пам'яті графічний драйвер і переводить адаптер у графічний режим роботи:

```
IniiCrapЩуаг СгарііОгіУЄГ: Іпіедег; уаг СгарИМосіе: Іпіедег; РаіИТоОгіУЄг:зігіпд);
```

Параметри процедури мають такий зміст: СгарііОгіуег - тип графічного драйвера, CrapHMocle — графічний режим роботи адаптера, РаіНТоОгі УЄГ — шлях до каталогу, де зберігаються файли ***.Ьді** (якщо не задане значення останнього параметра, пошук здійснюється у робочому каталозі програми). Зазначимо, що один Ь§і-файл може містити драйвери декількох типів. Для визначення типу драйвера в модулі дгарії оголошено константи. Деякі з цих оголошень наведемо нижче:

```
сопзі 0eieci=0: С6А=1: ЕСА=3: УСА=9;
```

Більшість дисплейних адаптерів може працювати в різних режимах. Потрібний режим роботи визначається параметром CrapiїMocle, значення якого для драйвера УСА задається такими константами:

```
сопзі
   У6АІо=0;   {640*200}
   У6АМесІ=1; {640*350}
   УСАНі =2;  {640*480}
```

Для автоматичного визначення графічного драйвера використовується константа Oeieci. У цьому разі процедура IniiCrapИ звертається до процедури йeieci-Сгарії:

```
OeiecіигарНОаг СгарНОгіуег, ЕгарїїМосіе: Іпіедег);
```

Процедура Реіесійгарії повертає значення графічного драйвера та графічного режиму для даного тину адаптера, передаючи їх процедурі Іпіідгарії. У разі збою

під час ініціалізації графічного режиму функція СгарііКезиІІ повертає код помилки. Наведемо коди типових помилок.

```
сопзі
CrOk              =0:    {помилок немає
ЗгІпПбгарИ        =1:    {не і ні ці алізовано графічний режим
СгНоіОеіесі       =-2;   {не визначено тип драйвера
(ЗгРі I eMcbPr nci =-3;   {не знайдено графічний драйвер
BrInyaIicIOriyer  =-4;   {некоректний тип драйвера
йгЕггог           =-11;  (загальна помилка
Crinyai i OPoLUcMиш =-14; {некоректний номер шрифту
```

Процедура CIozeCrapiп змінює графічний режим відеоадаптера на текстовий. Для тимчасового переходу з графічного режиму в текстовий використовується процедура КезіогеCriMocie, а повернення з текстового режиму у графічний здійснює процедура 5eiCrapHMocie(Moc!e: Іпіедег). Загальну схему роботи графічної програми демонструє приклад 5.1.

Приклад 5.1

```
изез дгарН;                {підключення графічного модуля}
yar дгсігіуег,             {графічний драйвер             }
    дгглосіе : іпіедег;    {графічний режим               }
Ьеділ
  дгсігіуег;=0eieci;       {визначити драйвер автоматично}
  Іпіідгарії(дгсігіУег.дгшос1е.''); {ініціалізація графіки}
  іТ СгарНКези1і=0 іНеп        {ініціалізація неуспішна}
    Ьеділ
        мгііе)п('помилка графіки'); Иа1і(1);
    епсі еізе                 {ініціалізація є успішною}
        Ьеділ
          {...}             (виклик графічних процедур}
          CIозебгарії;  {повернення до текстового режиму}
        епсі;
епсі.
```

5.2. Графічне вікно та система координат

Для виведення графічного зображення використовується координатний метод. Згідно з цим методом кожна точка на екрані задається двома прямокутними координатами. У режимі УСА лівий верхній кут екрана має координати (0,0), правий нижній - (639,479). Функції CeiMaxX та CeiMaxY повертають максимальні значення координат по горизонталі (А-) та по вертикалі (У) в поточному режимі роботи адаптера. Координати пікселя по горизонталі та вертикалі визначаються функціями беіХ та (МУ.

Система координат відеоадаптера має певні відмінності від декартової системи координат. По-перше, як уже зазначалося, вертикальні координати точок адаптера збільшуються зверху вниз. По-друге, координати пікселів можуть бути лише

цілочисловими. По-третє, зображення на екрані може розтягуватися. Якщо на екрані зобразити коло, в деяких відеорежимах воно виглядатиме як еліпс.

У загальному випадку формули перетворення декартових (*логічних*) координат (*ЙIX,eIУ)* в *екранні (X.¥)* враховують масштаб зображення, а також його горизонтальний і вертикальний зсув відносно лівого верхнього кута екрана:

$$mX = mX \text{ СПУ } хю<iX,$$
$$m¥- \quad НУд.\text{IУ} \quad МУ,$$
$$X = [\_ (IX \, x \, mX] + \text{AX},$$
$$У=i \quad ЛУх \quad mУ\backslash + ДУ.$$

Тут *mX, mУ* - масштаб по горизонталі і вертикалі; *хюX, НУ* - ширина і висота області зображення в екранних координатах; *хюдX, МУ* - ширина і висота області зображення в декартових координатах; *АX, А У* — зсув початку координат; I *x]* - ціла частина *x*.

Для обмеження області виведення зображення можна створити прямокутне графічне вікно:

5eiУieмРогі(x1, уі. x2. y2: Іпіедег; Сіір: Вооїеап);

Параметри xi, уі, x2, y2 процедури БеіУіемРогі є координатами лівого верхнього та правого нижнього кутів графічного вікна. Для відсікання зображення за межами графічного вікна параметру Сіір надається значення сі і роп, а для його продовження — значення cIipoTT.

Очищення графічного вікна здіїїснює процедура CIearУieиPorT, а очищення всього екрана та заповнення його кольором фону — процедура CIearOeyice.

5.3. Графічні процедури й функції

Наведемо перелік основних процедур і функцій бібліотеки ОгарИ (табл. 5.1). Для зручності всі підпрограми згруповані за функціональним призначенням.

Таблиця 5.1. Графічні процедури й функції

Призначення	Формат	**Параметри**
Керування курсором		
Поточні координати курсору	Типсііоп OeiX: іпіедег; Типсііоп СеіУ; іпіедег;	Відсутні
Максимальні координати курсору	Типсііоп бeiMaxX: іпіедег; Типсііоп EeiMaxУ: іпіедег;	Відсутні
Зсув курсору	ргосесііиге **МОУЄТОХ**, у: іпіедег);	х, у — координати точки
Графічні примітиви		
Дуга кола	ргосесііиге Аге (х, у : іпіедег; ЗіАпдІе. ЕпсІАпдІе, Касііиз; моrсі);	х, у — координати центру кола, БіАпдІе, ЕпсІАпдІе — початковий та кінцевий кути, Кас!иѕ — радіус

Призначення	Формат	Параметри
Заштрихований прямокутник	ргосесіиге Ваг(х1. уі. х2, у2: іпіедег);	хі, уі, х2, у2 — координати лівого верхнього та правого нижнього кутів
Коло	ргосесіиге Сігс!е(х,у: іпіедег; Касііиз: Иогсі);	х, у — координати центра; Касііиз — радіус кола
Многокутник	Ргосесіиге ОгамРоІуШитРоіпІз; когй; уаг РоїуРоіпІз);	МитРоі піз — кількість кутів, РоїуРоіпІз — масив координат вершин
Еліпс	ргосесіиге РіПЕПірзе(х, у: іпіедег; ХКасііиз, УКасііиз: могсі);	х, у — координати центра; Хгасііиз, Угасііиз — радіуси по осях х та у
Відрізок	ргосесіиге Ипе(х1, уі, х2, у2: іпіедег);	хі, уі, х2, у2 — координати двох кінців відрізка
Прямокутник	ргосесіиге Кес1апд1е(хі. уі, х2, у2: іпіедег);	хі, уі, х2, у2 — координати лівого верхнього та правого нижнього кутів
Відрізок, початок якого збігається з позицією курсору	ргосесіиге І_іпеТо(х,у: іпіедег);	х, у — координати кінця відрізка
Точка	ргосесіиге РиІРіхе1(х. у: іпіедег; Рі хе]: могсі);	х, у — координати точки; Рікеї — колір точки
Сектор еліпса	ргосесіиге ЗесІог(х, у: іпіедег; ЗІАпдІе, ЕпсІАпдІе, ХКасііиз, УКасііиз: могсі);	х, у — координати центра еліпса; ЗІАпдІе, ЕпсІАпдІе — початковий та кінцевий кути, ХКас1ііи5, УКасііиз — радіуси по вісях абсцис і ординат
Керування кольором і стилями		
Колір фона	Типсііоп 6еІВкСо1ог;игагсі;	Відсутні
Колір ліній	ТипсТіоп ЗеТСоїог;могсі;	Відсутні
Код максимального кольору	Типсііоп СеІМахСо1ог;могсІ;	Відсутні
Заповнення фігури кольором	ргосесіиге ПоосІРі 11 (х, у; іпіедег; Соїог; ТсоїогКеТ);	Х, у — координати точки всередині фігури; Соїог — колір, що обмежує фігуру
Встановлення кольору фону	ргосесіиге 5еІВкСо1ог(Со1ог; ТСоїогКеТ);	Соїог — колір фону
Встановлення кольору ліній	ргосесіиге 5еТСо1ог(Со1ог; ТСоїогКеТ);	Сої ог — колір контуру фігури
Встановлення стилю заповнення	ргосесіиге 5еТРі115Ту1е(РаІІегп; могсТ, Соїог; могсі):	Раїїегп — тип заповнення, Соїог — колір заповнення
Стиль лінії	ргосесіиге ЗеТИ пе3Ту 1 е(і_і пе3Ту 1 е; могсі; Раїїегп; *югб;* ТНіскпезз; дагсі);	І_і пе3Іуі е — стиль лінії (неперервна, пунктирна тощо); Раїїегп — допоміжний параметр; ТНіскпезз — товщина лінії

продовження &

Таблиця 5.1 (*продовження)*

Призначення	Формат	Параметри
Робота з пам'яттю		
Запис зображення в пам'ять	procesiиre ЄeIItade(x1. yi, x2, y2: iпiедег; yar BiiMap);	xi, yi, x2, y2 — координати лівого верхнього та правого нижнього кутів зображення; BiiMap — зображення в пам'яті
Виведення зображення з пам'яті на екран	procesiиre РиIItade(x, y: iпiедег; yar BiIMap; B I Ш I: мого);	x, y — координати лівого верхнього кута зображення; BiIMap — зображення в пам'яті; Bi IBI i — спосіб виведення зображення на екран
Розмір зображення в пам'яті	Типciiоп Iтаде$i2e(x1, yi, x2, y2: iпiедег): моrсі;	xi, yi, x2, y2 — координати лівого верхнього та правого нижнього кутів прямокутного зображення
Обробка тексту		
Встановлення стиля тексту	procesiиre 5eITex151y1e(РопI, йi recTioп; дагсі: СЇiагЗіге: моrсі);	Ропі — шрифт, Oireciiоп — напрям тексту, СЇiагЗire — розмір літер
Вирівнювання тексту	procesiиre 5eITexиизIiTy(НогI2, Уегі: УУОГСЗ);	Hori 2, Уегі — горизонтальний і вертикальний напрями виведення тексту
Висота тексту	Типciiоп TexiНеідіїКTexI3iгіпд: зігіпд): моrсі;	Texi3Iгіпд — рядок тексту
Ширина тексту	Типcii оп TexIЬ'i ciїii (TexI3Iгi пд: зігіпд): моrсі;	TexI3iгіпд — рядок тексту
Виведення тексту на екран	procesiиre ОиITexKTexI3Iгіпд: зігіпд);	Texi3Iгіпд — рядок тексту
Виведення тексту на екран у заданих координатах	procesiиre ОиiTexiXУ(x. y: iпiедег; Texi3Iгіпд: зігіпд);	x, y — координати зображення рядка, Texi3Iгіпд — рядок тексту

5.4. Керування кольором і стилями

Для зображення графічних об'єктів використовується кольорова палітра з кодами кольорів 0, i, ..., тахсоїогз. Загалом драйвер УСА дозволяє відображати до 2^{10} кольорових відтінків, але одночасно - не більше 16. Тому значення константи тахсоїогз дорівнює 15 і за замовчуванням використовуються 16 кольорів, перелічених нижче.

```
Сопзі
Віаск = 0; {чорний}          Віие  = 1; {темно-синій}
Сгееп = 2; {темно-зелений}   Суап = 3; {темно-блакитний}
КecI = 4; {бордовий}         Мадепіа = 5; {темно-фіолетовий}
Вгомп = 6; {рудий}           ИдIїidгау = 7; {сірий}
```

```
Оагкдгау = 8; {темно-сірий}  IідіїіВИие = 9; {СИНІЙ}
Идііідгееп = 10; {зелений}   Идіїісуап = 11; {блакитний}
ИдНігесі = 12; {черзоний}    Ілдіїітадепіа = 13; {фіолетовий}
УеПом = 14; {жовтий}         МНіїе = 15; {білий}
```

Використання інших кольорових відтінків вимагає доволі складної техніки керування кольоровими палітрами, але її розгляд не належить до кола завдань даного підручника. Синтаксис процедур і функцій, що встановлюють кольори ліній і фону, наведено в табл. 5.1. Зазначимо, що установка кольору впливає на всі подальші виклики процедур зображення геометричних об'єктів. За замовчуванням вибирається чорний фон і білий колір ліній.

Для зафарбування геометричної фігури використовуються процедури заповнення замкненого контуру заданим кольором у певному стилі (див. табл. 5.1). Наведемо оголошення констант, що визначають стиль заповнення, а також стиль і товщину ліній.

```
Сопзі {стиль заповнення}
ЕтріуРі11 = 0; ЗоІісіРіП = 1; ЦіпеРп11 = 2; иЗІазНРіП = 3; ЗІазіїРіП = 4;
               ВкЗІазіїРіП = 5; ИВкЗІазіїРіІІ = 6;
НаісіїРіІІ = 7; ХНаісіїРіП = 8; ІпІегІеауеРіП - 9; ИісіеОоіРіП = 10; СІозеРоіРіП = 11;
               ІІзегРі11 = 12;

Сопзі {стиль ЛІНІЙ}
Зо1ісИп=0; 0оііесИп=1; Сепіегі.п=2;
0азИесИп=3; ІізегВіНп^;

Сопзі {товщина ліній}
МогтЬІісШі=l; ТМскУісШКЗ:
```

5.5. Графічні примітиви

Графічні примітиви — це геометричні фігури, що їх можна відобразити на екрані за допомогою окремих процедур.

Приклад 5.2

Розглянемо програму, яка зображує концентричні кола випадково вибраними кольорами (рис. 5.1). Діаметр кіл спочатку збільшується, а згодом - зменшується. Після зображення кіл програма виводить відрізки прямих, що мають випадковий колір і розташування. Потім зображуються прямокутник, еліпс, сектор еліпса і трикутник.

```
Ргодгат ех5_1;          {відображення графічних примітивів}
изез дгарИ.сгі;
уаг сіг.шосіе.і:іпіедег;
Ьеділ
  гапсіотіге;    {ініціалізація генератора випадкових чисел}
  сіг;=0еіесі;
  ІпіібгаріїСсіг.ітюсІе."с;\Ьр\Ьді');
             {концентричні кола, радіус яких збільшується}
```

```
  Tor i :=1 To 480 cio  {максимальний радіус - 480 пікселів}
  Ьеділ
    5eTCo1or(гапсіопі(15));                    {вибір кольору}
    CircIeCдеїшахХ CIIУ 2, деТшахУ CIIУ 2.i);
    cie1ay(200);                               {затримка відображення}
  епсі;
                {концентричні кола, радіус яких зменшується}
  Tor i: =480 сіомпТо 1 cio
  Ьеділ
    5eiCo1or(гапсіот(15));
    Circie(дєТтахХ CIIУ 2. деТшахУ CIIУ 2,I ) ;cie!ay(200);
    5eTCo1or(0);          {колір фону для приховування кола}
    CircIeCдеТшахХ CIIУ 2, деТшахУ CIIУ 2,i);cie1ay(200);
  епсі;
  Tor i :=1 To 480 cio                                  {лінії}
  Ьеділ
    ЗеТСоІ or(гапйотС 15));  сієї ay(200);
    Ипе(гапсіот(640) ,гапсіогп(480) ,гапсіоггК640) ,гапсіот(480));
  епсі;
  геасіїп;
  CIearOeyice;                                  {очистка екрана}
  ЗеТВкСоїог(15);   5eTCo1or(4);
  КесТапдІе(50,40,100,200);                     {прямокутник   }
  5eTCo1or(1);
  Pi11E11ip5e(6eTMaxX CIIУ 2,200, 150, 50);     {еліпс    }
  5eTCo1or(12); 5есТог(400,100,0,270,50.70);    {сектор   }
  5eTCo1or(10); 1іпе(140,200,250,50);           {трикутник}
  II пе(250,50,350,120);  Ипе(140.200,350,120);
  геасіїп;
епсі.
```

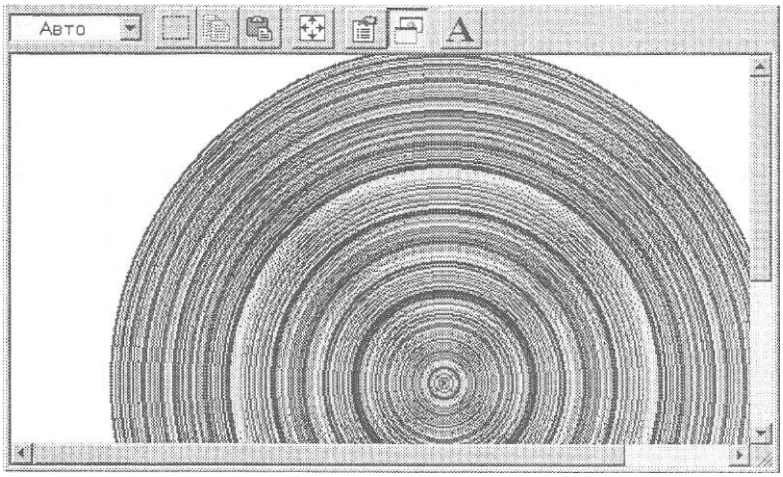


Рис. 5.1. Результати роботи програми ex5_1.
Відображення концентричних кіл

5.6. Зображення текстової інформації у графічному режимі

Текст у графічному режимі виводиться з використанням шрифтів, що зберігаються у файлах із розширенням сїїг. Параметрами шрифтів визначається розмір літер, напрям та спосіб вирівнювання тексту тощо. Значенням цих параметрів зіставлені такі константи:

```
Сопзі: {типи шрифтів}
ОеГаиПРопI = 0: ТгірIехРопI = 1: ЗтаПРопі = 2:
5апз5егі ТРопі = 3: СоїИісРоггЬ = 4;

Сопзі          {напрям тексту}
Ногігйіг=0:    {зліва направо}
УегШг=1;       {знизу догори }

Сопзі          {вирівнювання тексту}
IeTITexi  =0; СепіегТехі=1; КідіНТехі=2;   {горизонтальне}
ВоїіошТехї=0: СепіегТехI=1; ТорТехї=2:     {вертикальне  }
```

Усі шрифти, крім шрифту ОеТаиПРопІ, є векторними, тобто їх елементи формуються як сукупність векторів, що характеризуються напрямом і довжиною. Жоден шрифт, крім зазначеного, не підтримує кирилиці - для виведення її символів шрифти слід модифікувати. Власне виведення тексту виконують процедури іМТехі і ОиїТехШ, які було наведено в табл. 5.1.

5.7. Побудова графіків функцій

Технологію побудови графіків функцій продемонструємо на прикладі функції $/(x) = |\sin^{} + \cos|x|$. Графік функції побудуємо за точками *(x,/(x)):* циклічно задамо ряд значень абсциси {*,}, обчислимо відповідні значення ординати $y_1 = /(x_i)$ і зобразимо серію відрізків, що з'єднуватимуть точки г/;_і) із точками (x_i, yi).

Для зображення графіка слід перевести логічні координати його точок у їх екранні еквіваленти. Область визначення розглядуваної функції - це вся вісь абсцис, а область значень — відрізок [-2; 2]. Відрізок логічної вісі ординат завдовжки 4 слід розтягнути у вертикальний екранний відрізок більшої довжини, тобто домножити на коефіцієнт, що його називатимемо масштабом. Вважатимемо, що масштаб є цілим числом від 20 до 100. З урахуванням того, що центр логічної системи координат збігається із центром екрана, а також того, що напрям екранної вісі ординат є зворотним до напряму логічної вісі ординат, отримаємо таку формулу для обчислення екранної ординати *у_екрана* за логічною ординатою */(x):*

$$y\_екрана := y\_центра^{}екрана - масштаб \cdot /(x);$$

Яким чином визначати логічні абсциси? Найкраща якість зображення графіка досягається тоді, коли кожній екранній абсцисі відповідає логічна абсциса. Наприклад, якщо вісь абсцис проходитиме посередині екрана, який містить 480 точок

по вертикалі, то екранний піксел (0,240) буде зіставлений з логічною точкою (л^.О), піксел (1,240) - з точкою (x_2,0). Такої відповідності легко досягти, якщо перерахувати екранну координату *x_екрана* у відповідну логічну координату *x*. Це можна зробити за такою формулою:

x := (x_екрана - x_центра_екрана) / масштаб;

Приклад 5.3

Програма ex5_2 будує графік функції /(x) = |зіп *x*\ + соз|x|. Значення масштабу вводиться до змінної зсаіе, а координати центра екрана на початку роботи програми присвоюються змінним сх і су. Графік будується процедурою Ьии1сі_дгарii. Дві інші процедури зображують координатні вісі і розмічають їх. При розмітці осей координат використовуються рядки тексту, обробка яких детально розглядатиметься в розділі 7.3. Результати роботи програми наведено на рис. 5.2.

```
Ргодгат ех5_2;              {графік функції у=|зіпх|+соз|х|}
изез сгІ.дгарИ:
уаг сіг,тосіе:іпіедег;          {графічний драйвер і режим }
    зса1е:геа1;                 {масштаб                   }

    сх,су:іпіедег;              {координати центра екрана  }
{===== аналітична функція, графік якої будується =======}
іипсііоп Г(х:геа1):геа1;
Ьеді п
  і:=аЬз(зіп(х))+соз(аЬз(х));
єпсі;
{============ виведення вісей координат ==============}
ргосесііиге ахез;
Ьеді п
  ЗеІСоі ОГ(1);
  ИпеССх.О.сх.ОеІМахУ ): {вісь У}
  І_іпе(0,су,6еІМахХ,су);  {вісь Х}
                                          {стрілки на вісі Х}
  Ипе(6еІМахХ-10,(су)-5.6еІМахХ,су);
  І-іпеССеІМахХ-ІО, (су)+5.йеІМахХ,су);
                                          {стрілки на вісі V}
  Ипе(сх,0,(сх)-5,10);
  Ипе(сх.0.(сх)+5,10);
                                          {написи на вісях  }
  оиІІехІху(2еІМахХ-10,су+20, 'Х'};
  оиІІехІху(сх+10.10, 'У');
епсі:
{==^========;== розмітка вісей координат ==============}
ргосесііиге дгасііиіг;
уаг і,1;геа1;                          {позначки на вісях}
     з;зІгіпд[7]; {рядок для виведення позначки на вісях}
Ьеді п
  ЗеІСоІогШ;
  1:= 0;                {позначка '0' в центрі координат}
```

```
{                    розмітити ВІСЬ X                    }
{позначки у вигляді вертикальних рисок }
  гереаї
    i:=cx+(-з)*зca1e;
    1 іпє(гоипсI(i). су-5, гоипсі(i) ,су+5);
    іТ ,J<>0 іііеп       {не виводити повторно позначку '0'}
    Ьеділ
      5їг(-з: б: 2, з);  {перетворити числову позначку
                  на текстову і відобразити її на екрані}
      СМТехШ( гоипсі(i) -10. су+10.5);
    єпсі;
{зобразити позначки на горизонтальній вісі}
    i:=cx+;i*5ca1e;
    Ипе( гоипсі (i). су-5, гоипсі (i) ,су+5);
    і і 1<>0 їЬеп
    Ьеділ
      з і г у :б;2.з);          {перетворення числа на рядок}
      (МТехШ(гоипсI(i)-IO,су+10,  з):
    епсі:
    1:=1+pi:
 ипііі ,i=8*pi;                  {позначки вичерпано}
(                    розмітити вісь У                    }
  1 ; = 0 ;
  гереаї
    i:=су-,]*5ca1e;       {розташування позначки на вісі У}
    1іпе(сх+3,гоипсі(i) .сх-3, гоипсі (i));
    іТ л<>0 їііеп
    Ьеділ
      з1г(1:4:1.з);          {перетворення числа на рядок}
                  {вивести значення, відповідне позначці
                 на вертикальній вісі в межах від 0 до 2}
      ОиїТехІХУ (сх+15, гоипсі (i) - 2, з);
    єпсі;
    i:=су+з*зca1є:
    1іпе(сх+3,гоипсі(i), сх-3, гоипсі (i)):
    їТ 1<>0 їНеп
    Ьеділ
       зіг(-;i :4:1,з);         {перетворення числа на рядок}
                  {вивести позначки на
                   вертикальній вісі в межах від -2 до 0}
       ОиїТехШ(сх+15, гоипсі (i) - 2, з);
    епсі;
    1:=1+0.5;
                   {розмітити вертикальну вісь через
                    кожні 0.5 ділень                    }
  ипїіі ^=2:       {діапазон позначок: від -2 до 2      }
 епсі:
{================ побудова графіка ==================}
ргосесііиге Ьиі1сі_дгарИ;
уаг
  хО.уО.хI.уI:іпіедег;  {екранні координати кінців
                        відрізка графіка               }
  x:rea!;
```

```
Ьеділ
  Тог хО :=0 То СеїМахХ сіо          {вибір екранної абсциси}
  Ьеділ
    х:=(х0-сх)/зса1е;  {перерахувати пікселі екрана
                    в логічні координати абсциси графіка}
    уО:=гоипсі(су-5са1е*Т(х));
    іТ х0>0 їііеп
      Iіпе(хО.уО.хI.уI):       {зобразити відрізок графіка}
    х1:=х0;                    {зберегти координати кінця
    у1:=у0;                      попереднього відрізку     }
  епсі;
епсі;
{================ основна програма =•»======-======}
Ьеділ
  сіг;=ОеТесТ;                 {визначити графічний драйвер}
  ІтТ6гарИ(сіг,тосіе.'с3:\Ьр\Ьді');   {ініціалізувати
                                       графічний режим     }
  сх:=СеТМахХ сііу 2;          {визначити координати       }
  су:=йеТМахУ сііу 2:          {центра екрана              }
  5еТВкСо!ог(15);              {визначити колір фону       }
  5еТСо1ог(1);                 {визначити колір ліній      }
  ОиТТехТСіприТ зсаіе (20 ... 100):');
  доТоху(27.1);
  геасКзсаІе):                 {увести масштаб зображення }
  СІеагОеуіce:                 {очистити екран             }
  ахез;                        {зобразити вісі координат   }
  дгасіїііг:                   {розмітити вісі             }
  ОиТТехТХУ(10,30,'у=|зіп(х)|+со5|х|'); {вивести напис   }
  Ьиі1сі_дгарН;                {зобразити графік           }
епсі.
```

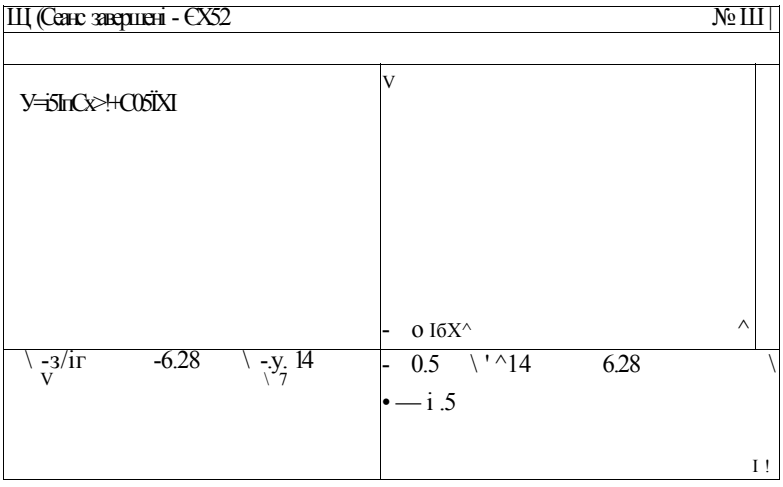


Рис. 5.2. Результати роботи програми ех5_2.
Графік функції у = |зіпх|+соз|х| у масштабі
30 пікселів на одну логічну одиницю

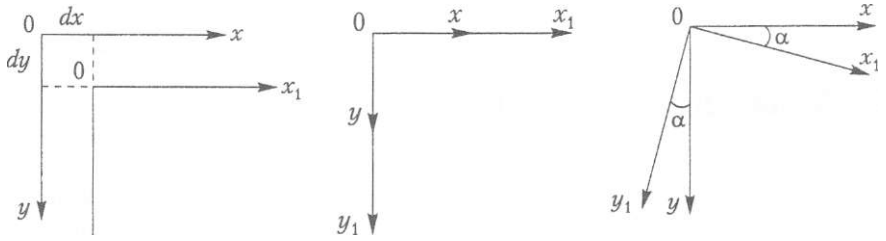
5.8. Перетворення координат ї об'єктів

У комп'ютерній графіці перетворення координат застосовують для зміни масштабу зображення, отримання симетричних частин графічних об'єктів, дублювання частин зображення тощо. Перетворення координат об'єкта — це їх переобчислення за певними формулами. У подальшому розглядатимемо лише лінійні перетворення координат на площині, під час яких зберігаються такі геометричні властивості об'єктів: прямі лінії залишаються прямими, зберігається паралельність прямих, а також відношення площ геометричних фігур. Основними лінійними перетвореннями є паралельне перенесення, розтягування (стискання) і поворот геометричних об'єктів.

Припустимо, що у декартовій системі координат задана плоска геометрична фігура, *(x,y)* — координати однієї з її точок. Нехай (хі,г/і) - координати тієї самої точки фігури після їх перетворення. Наведемо формули паралельного перенесення геометричної фігури на *сіх* одиниць уздовж осі *X* і на *сіу* одиниць уздовж осі *У* (формула *а),* розтягування уздовж осі X у *кх* разів і вздовж осі *Уку* разів (формула *б)* і повороту навколо початку координат на кут а (формула *в).*

Лінійні перетворення проілюстровано на рис. 5.3. Необхідно зазначити, що будь-яке перетворення координат об'єкта можна розглядати як перетворення координатних осей.

а) $\begin{cases} Гх, = x \pm cbc; \\ [y_{\ell} = y \pm ay, \end{cases}$ *б)* $\begin{cases} (x_{\gamma} = x\text{-}к_{x}; \\ yУ\ = y\text{-}к_{y}\ \end{cases}$ *в)* $\begin{cases} Гх_{\gamma} = х \bullet соз\ а + y\ \text{-}зіпа; \\ [х_4 = -х\text{-}зіпа + y \bullet соз\ а. \end{cases}$



*У і**

а *б* *в*

Рис. 5.3. Лінійні перетворення координат: паралельне перенесення (а); розтягування або стискання (б); поворот навколо початку координат (в)

Приклад 5.4

Розробимо програму обертання відрізка навколо одного зі своїх кінців. Вважатимемо, що початок логічних координат збігається із нерухомим кінцем відрізка, а довжина відрізка дорівнює *г.* Тоді рухомий кінець відрізка матиме спочатку координати (г,0), а після обертання на кут а — координати (г соз а + 0 • зіп а, - *г* зіп а + + 0 • соз а) = *(г* соз а, - *г* зіп а). Нерухомим кінцем відрізка вважатимемо центр екрана, і з урахуванням того, що до цієї точки було перенесено початок координат, а також того, що екранна вісь ординат спрямована донизу, отримаємо такі формули

координат рухомого кінця: (x_c + r соз а, y_c + r зіп а), де (x_c, y_c) — координати центра екрана.

```
Рrодгат ех5_3;
изез сгї.дгарЬ;
уаг сі.ш.              {графічний драйвер і графічний режим}
    х.у.               {координати рухомого кінця          }
    г-.іпїедег:        {довжина відрізка                   }
    а:геа1;            {кут повороту відрізка              }
Ьеділ
  сі:=сіеїесї;         {ініціалізація графічного режиму    }
  іпі1дгарії(сі,іті, 'сІ:\Ьр\Ьді');
  г:=100:              {ініціалізувати довжину відрізка    }
  зеїЬксо1ог(15);      {колір фону                         }
  зеТсо"1ог(3);        {колір літер та ліній               }
  5еїїехїзїу1е(0,0.1):              {стиль літер          }
  оиїїехїху(100,50, 'Коїаїіопаї сіізріасетепї оТ Ііпе');
  оиїїехїху(100,80,'Ргезз апу кеу То ехії ');
  а:=0;
  гереаї
    х:=320+гоипсКсоз(а)*г);     {перетворення координат   }
    у: =240-гоипсКзіп(а)*г);    {рухомого кінця           }
    зеїсоіог(6);                {відображення відрізка    }
    1іпе(320,240.х,у);
    сІе1ау(2000);               {затримка                 }
    зеїсо!ог(15);               {затирання відрізка       }
    1іпе(320,240,х,у);
    а:=а+рі/36;                 {збільшення кута повороту }
  ипїіі  кеургеззєс!;
епсі.
```

5.9. Анімаційні ефекти

Під час розробки анімаційних програм потрібно створювати ефект пересування графічних об'єктів. Найпростіший спосіб реалізації цього ефекту полягає в тому, щоб намалювати зображення певним кольором, а потім приховати його шляхом повторного малювання в тих самих графічних координатах кольором фону. Наступного разу зображення відтворюється вже в нових координатах. Великі за розміром зображення пересуватимуться у такий спосіб надто повільно. Значно вищої швидкості, а отже і якості анімації можна досягти за допомогою копіювання зображень в оперативну пам'ять і виведення на екран їх копій у нових графічних координатах. Оскільки копія зображення зберігається в оперативній пам'яті, можливе її багаторазове виведення.

Процедури і функції, що застосовуються для створення анімаційних ефектів, наведені у табл. 5.1. Процедура СеїІтаде(х1, уі, х2, у2: ІпТедег; уаг ВіїМар) зберігає в пам'яті зображення, оточене прямокутником, лівим верхнім кутом якого є точка (х1,у1), а правим нижнім - точка (х2,у2). Параметр ВіїМар — це динамічна змінна, що в ній зберігатиметься зображення. Покажчик на змінну ВіїМар, як правило, нетипізований. На екран зображення виводиться за допомогою процедури

:'ИЇІтаде(х. у: Іпіедег; уаг ВіїМар; ВіїВН: MorcI). Координатами лівої верхньої точки зображення є значення параметрів x і y. Параметр ВіЇВ1 і визначає спосіб взаємодії нової копії зображення із зображенням, гцо вже є на екрані. Взаємодія здійснюється за допомогою логічних операцій, які застосовуються до кожного біта копії та оригіналу зображення. Значенням параметра ВіїВТї є одна із таких констант:

```
сопзї
№гта1Риї=0;     {Заміна наявного зображення на копію}
ХогРиї=1;       {Операція виключного "або"}
0гРиї=2;        (Об'єднувальне "або"}
АпбРиї=3;       {Логічне "і"}
і.оїРиї=4;      {Інверсія зображення}
```

Логічні операції над зображенням дають такі результати. У випадку використання константи МогтаІРиї зображення на екрані знищується і на цьому місці відображається копія з оперативної пам'яті. В разі вибору константи і-іоїРиї буде виведено копію зображення в інверсному коді: так, код міііїе (15_{10} = ПІІ2) замінюється кодом Ьіаск (0_{10} = 0000_2), код гесі (4_{10} = 0100_2) - кодом ІідЬїСуап (11_{10} = = 1011_2). Використання константи ХогРиї дає можливість видаляти зображення в тих точках екрана, де розміщено його оригінал. Якщо цю операцію застосувати двічі до тих самих координат екрана, то зображення не зміниться.

Приклад 5.5

Створимо програму, що імітує рух місяця на зоряному небі. Коли зображення місяця сягає межі екрана, напрям його руху змінюється. Один із можливих станів екрана під час роботи програми зображено на рис. 5.4.

```
Ргодгат ех5_4;          {місяць на небі, зіроньки сяють }
изез сгї.дгарії;
уаг сіг,гіі:іпїедег;       {графічний драйвер і режим       }
   х.у.                 {координати об'єкта під час руху}
   сіх.сіу:іпїедег;     {приріст координат               }
   рїг:роіпїег;         {покажчик на область пам'яті.
                          де зберігатиметься зображення  }
   5іге:Іпіедег;        {розмір необхідної для
                          збереження зображення пам'яті  }
   ІеТїХ.ІеШ,           {координати лівого верхнього і  }
   гідНїХ.гідІїїУ:іпїедег; {правого нижнього кутів
                            прямокутника, в який
                            вписано зображення           }
   і:іпіедег;              {параметр циклу               }
{============ відображення зоряного неба
ргосесіиге 5ку;
Ьеділ
  гапсіотіге: {ініціалізація генератора випадкових чисел}
  5еїВкСо1ог(1);               {визначення кольору фону}
  3еїСоїог(14);                {колір зірок             }
  Тог і:=1 їо 200 сіо          {відображення зірок      }
    Сігс1е(гапсІош(640), гапбот(480) ,1);
епсі:
```

```
{============= відображення місяця на небі ============}
ргосесіиге Иоол;
Ьеділ
  5еICo1ог(14);
  Аге(450.100.270,90,50);
  Аге(390,100,320,40,80);
  5еїРі1151у1е(1,14);
  ПоосіРі 11 (480,100,14);
епсі;
{====== збереження зображення у динамічній пам'яті ======}
ргосесіиге ЗауеСІір;
Ьеділ
  1еШ:=445;                    {координати прямокутника,}
  1еШ:=45;                     {у який вписано місяць    }
  гідШ:=505;
  гідШ:=155:
  5І2е:=ітадезі2е(1еШ ЛеШ.гідШ.гідІїіУ);
  деІшеш(рІг,5і2е);                      {виділити пам'ять}
  деїітадеО е Ш, 1 е Ш, гі д Ш. гі д Ш. р і г");
                            {зберегти зображення в пам'яті}
  риїітаде( 1 е Ш. 1 еТІУ,ріг^A,хогриі); {приховати зображення}
єПСІ;

ргосесіиге НОУЄ;
Ьеділ
  х:=1еШ; у:=1еШ;           {стартові координати      }
  сіх:=10;сіу:=10;          {приріст координат        }
  гереаі:                   {зсув зображення          }
    х:=х+с!х;               {зміна координат місяця   }
    у: =у+сіу;
    риІішаде(Х.У,ріг^A,хогри1;);    {зобразити фігуру
                                      в нових координатах   }
    сіе!ау(2000);                    {затримати рух         }
    ри1ішаде(Х,У.р1:г^A,хогриі);    {сховати фігуру
                                      в старих координатах  }
    іТ (х>640) ог (х<0) {якщо фігура сягнула межі екрана,}
      ТТїеп сіх:=-сіх       {змінити напрям її руху       }
      еізе
        IT (у<0) ог (у>480) ТНеп сіу:=-сіу;
  ипііі кеургеззесі;
епсі;
{=================== основна програма ==================}
Ьеділ
  сіг: =0еїес1;
  IпііСгарії(сіг.ш,'');
  5ку;
  Мооп;
  ЗауеСІі р;
  НОУЄ;
епсі.
```

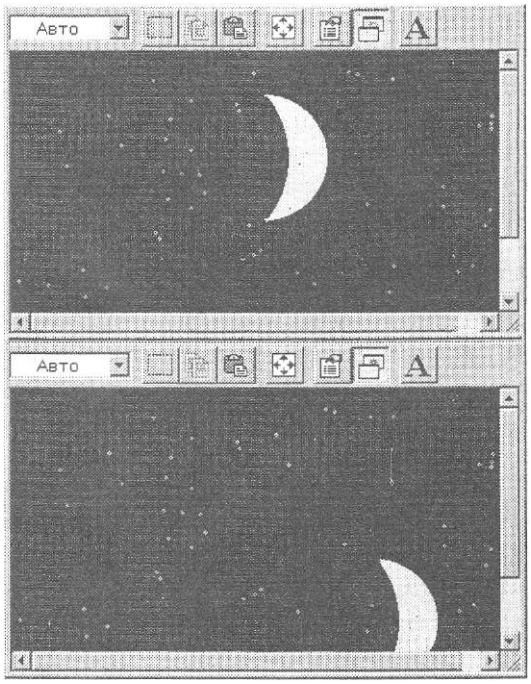


Рис. 5.4. Результати роботи програми ex5_4. Рух місяця на небі

5.10. Фрактальні зображення

Фундатор напряму фрактальної геометрії Б. Мандельброт дав таке визначення фрактального зображення, або фрактала (від англ. ігасііоп - дріб): «Фракталом називається структура, що складається з частин, які подібні цілому». Можна взяти певну частину ідеального фрактала, збільшити її в будь-яку кількість разів, і вона в точності повторюватиме вихідний об'єкт або певну його частину. Візерунок, який зображено на рис. 5.5, називається множиною Мандельброта і є одним з найвідоміших фрактальних об'єктів.

Множина Мандельброта — представник групи фракталів, що називаються алгебричними, оскільки їх структура визначається алгебричними формулами. При побудові таких фракталів вхідні дані послідовно перетворюються за правилом, заданим цими формулами, і результати кожного наступного перетворення залежать від результатів, отриманих під час виконання попереднього. Залежно від початкових умов функція, що описує таку систему перетворень, може наблизитися до нескінченності, збігтися до певного скінченного числа (числового діапазону) або нескінченно варіюватися у певному діапазоні.

Множина Мандельброта визначається таким рівнянням:

Тут змінна *2* і параметр *С* — комплексні числа, *п* — номер ітерації. Нагадаємо, що кожне комплексне число можна подати у вигляді *тю = а + Ьі,* де *а* і *Ь* — дійсні числа, *і* — уявна одиниця, тобто число, що задовольняє умову $i^2 = -1$. Дійсні числа *а* = = Ке(да) і *Ь* = Іт(да) називаються відповідно дійсною і уявною частинами комплексного числа *тю.* Піднесемо комплексне число до квадрата: $тю^2 = (а + Ьі)\text{-}(а + Ьі) = = а + аЬі + аЬі + і^2Ь^2 = а^2 + 2аЬі - Ь^2$. Застосувавши цю формулу до рівняння, що описує множину Мандельброта, отримаємо

$$Ке(2_п) = (К е \wedge і))^2 - (Ц \wedge - і))^2 + Ке(С);$$
$$Іш(2_{„}) = 2Ее(2_{л\_1})Іт(2_{„\_1}) + Іт(С).$$

Комплексне число зручно зображувати точкою на площині, абсциса й ордината якої відповідають дійсній та уявній частині числа. Щоб отримати зображення множини Мандельброта, потрібно виконати певну кількість ітерацій за визначеними вище формулами для кожної точки т прямокутника, лівим нижнім кутом якого є точка (-2; -1,25), а правим верхнім - точка (1,25; 1,25). Ітерації тривають доти, доки не стане істинною умова $І 2_п \setminus > 2$ або доки не буде виконано певну кількість ітерацій. При цьому числом ітерацій визначається колір точки *тю,* а дійсна та уявна частини комплексної константи *С* дорівнюють відповідним координатам *ж.* Ке(С) = *7ю*$_х$, Іш(С) = *тю*$_у$. Зауважимо, що модуль комплексного числа дорівнює квадратному кореню з суми квадратів його дійсної та уявної частин: $І ш\setminus = 7ке^2(да) + 1т^2(да)$.

Приклад 5.6

Програма ех5_5 будує множину Мандельброта (рис. 5.5). Координати всіх точок екрана перетворюються так, щоб екран став зображенням прямокутника, лівим нижнім кутом якого є точка (-2; -1,25), а правим верхнім - точка (1,25; 1,25).

```
ргодгат ех5_5:
изез дгарИ;
сопзі тіпх = -2:          {координати лівого нижнього кута  }
      шіпу = -1.25:       {прямокутної множини точок        }
      таxх - 1.25;        {координати правого верхнього кута}
      шаху = 1.25;        {прямокутної множини точок        }
      5сгеепМісііІі=640:  {кількість пікселів на екрані     }
      5сгеепНеідііі=480;
уаг сіх.сіу:геаі;         {приріст координат пікселів       }
    х.у,соїог:іпіедег;{поточні координати та колір піксела}
    сігі УСГ . шосіе: і пїедег; {графі чний драйвер і режим  }
|============ визначення кольору піксела ================}
Типсіі оп Са1с_рі хеі(геС л тС:геа1):і піедег;
{параметри - дійсна та уявна частини комплексного числа}
сопзі
  шах_ііегаііоп=128;                    {кількість ітерацій}

уаг
  о!сі_а:геа1:           {попереднє значення дійсної частини}
  ііегаііоп:іпіедег;      {лічильник ітерацій               }
  а.Ь:геаі:      {дійсна і уявна частини комплексного числа}
```

```
  z:rea1:        {довжина вектора z                    }
Ьеdіп
  a:=0; Ь:=0; iieraiion:=0;
  гереаї:
    o1ci_a:=a;              {запам'ятати попереднє значення}
    a:= a*a - Ь*Ь + reC;        {нова дійсна частина      }
    Ь:= 2*o1c1_a*Ь + ішС;       {нова уявна частина       }
    iIeraiion:=iieraiion+1;     {перейти до наступної
                                 ітерації                 }
    r: = a*a + Ь*Ь:             {квадрат модуля числа     }
  иniii (2 > 4) or (iieraiion > max_iieraiion);
  Ca1c_pixe1:=iieraiion;
epci:
|================ основна програма ====================}
Ьеdіп
 CIГIУЄГ:=CIЄIЄCI;
 iniidгapИ(c!riУег,тос1е,'');
 cix      (maxx-tipx)/5creepИic1iИ;
 c1y:= (maxy-tipy)/5creepHeidИi;
 Tor y:=0 io 5creepHeidHi-1 cio
   Tor x:=0 io 5creepMicIiIi-1 cio
   Ьеdіп
     coї or: =Ca1 c_pi xei (tipx+x*c1x, shipy+y*c1y);
     puipixeïCx,y,coїor);
   epci;
epci.
```

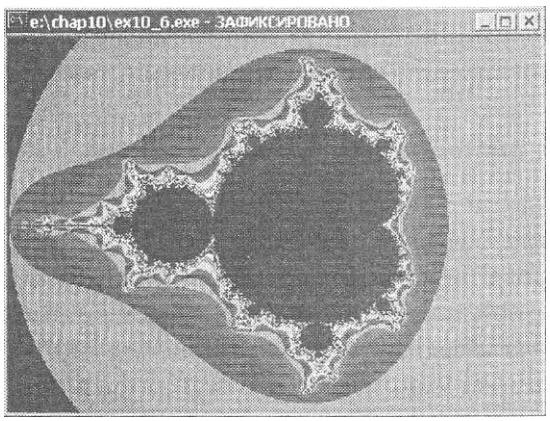


РИС. 5.5. Результати роботи програми ex5_5. Множина Мандельброта

Висновки

- Програма, що працює у графічному режимі, використовує графічні драйвери — файли, які містять інформацію про властивості відеоадаптерів. У середовищі Borlandpcs RascaI 7.0 графічні драйвери зберігаються у файлах з розширенням Ьді.

- Бібліотека графічних підпрограм міститься в модулі СгарЬ, записаному у файлі **...\ипіі5\дгарЬ.іри.**
- Для використання графічних засобів комп'ютера слід ініціалізувати графічний режим роботи дисплейного адаптера.

4- У графічному режимі кількість пікселів на екрані визначається роздільною здатністю відеоадаптера і дисплея. Роздільна здатність адаптерів УСА становить 640х480 пікселів. Лівий верхній кут екрана має координати (0,0), правий нижній - (639,479).

- Для зображення графічних об'єктів використовується кольорова палітра, що містить кольори, які кодуються цілочисловими значеннями 0, 1, ..., тахсоїог. На адаптері УСА одночасно може відображатися до 16 кольорів.
- Для виведення тексту в графічному режимі використовуються шрифти, записані у файлах з розширенням **сііг.**

+ При відображенні геометричних об'єктів використовуються формули перетворення логічних координат об'єкта на екранні. Ці формули враховують зміни масштабу об'єкта і зсув початку координат.

- Для зсуву, повороту, стискання або розтягування геометричного об'єкта застосовуються лінійні перетворення його координат. Лінійні перетворення характеризуються такими властивостями: прямі лінії залишаються прямими, паралельність прямих і пропорційність відстаней, а також відношення площ геометричних фігур зберігаються.

+ Анімація об'єктів здійснюється копіюванням зображення в оперативну пам'ять і виведенням його копії на екран у нових координатах.

- Фракталом називається структура, що складається з частин, подібних до цілого.

Контрольні запитання та завдання

1. Для чого призначені графічні драйвери?
2. Як ініціалізувати графічний режим?
3. Скільки кольорових відтінків можна відобразити за допомогою адаптера УСА?
4. Наведіть формули перетворення логічних координат на екранні.
5. Які перетворення координат об'єктів називаються лінійними?
6. Як відобразити текст у графічному режимі?
7. Як реалізувати анімацію?
8. Дайте визначення фрактального зображення.

Вправи

1. Процедура_____ініціалізує графічний режим роботи відеоадаптера.
2. Роздільна здатність відеоадаптера — це_____.
3. Пікселі у лівому верхньому і правому нижньому кутах екрана мають координати_____і_____ відповідно.

4. Графічна вісь ординат спрямована_____.
5. Для відображення кола використовується процедура_____, прямокутник відображається процедурою_____, відрізок прямої можна вивести за допомогою процедури_____, точка відображається процедурою_____.
6. Колір фону встановлює процедура_____, а колір ліній - процедура

7. Виведення тексту у графічному режимі здійснюється процедурою

8. Для створення анімаційних програм використовують процедури

9. Розмір зображення в оперативній пам'яті визначає функція__
10. Виведення зображення на екран із оперативної пам'яті здійснюється процедурою_____.
11. Логічна операція_____знищує наявне на екрані зображення під час виведення з пам'яті нового зображення.
12. Знайдіть помилку у фрагменті програми, що зображує еліпс, який рухається,

```
x:=100;
БеТВкСоїогЦ);
гереаї;
   5еїСо1ог(15):
   3еТТРі1151у 1е(1,15);
   РіПЕПірзе(x,100.80,50);
   OelayCЮOO):
   3еТСоіогЦ);
   5еїРіП5іу1е(1.15);
   РіПЕПірзе(x,100,80,50);
   IT x>640 ТИеп x:=x-10;
ипїіі кеургеззесі:
```

Задачі

1. дослідити область визначення і побудувати графік функції $y = (x + 3) / (x - 2)$.
2. Побудувати на екрані множину точок, координати яких задовольняють таку нерівність: $x^2 + y^2 < 2(|x| + |y|)$.
3. Зобразити криву «равлик Паскаля», що означена в полярних координатах таким рівнянням: р = а соз <р + /, де I - стала величина, кут <р змінюється від фО до *фО +2п.*
4. Зобразити астроїду, що визначена такими рівняннями в параметричній формі: $x = K\cos^3 \phi$, $y = i?\sin^3 \phi$, де K — стала величина, кут ср змінюється від 0 до 2л.
5. Зобразити на екрані квадрат, що рухається периметром нерухомого трикутника і одночасно обертається навколо своєї вершини.
6. Зобразити на екрані календар, аркуші якого відриваються і розлітаються, а на нових календарних аркушах змінюються числа і текст.

Розділ 6

Методології розробки програм

4- Метод низхідного проектування

• Принципи модульного програмування

4- Методи структурування програм

4 Конструювання модулів у мові Pascal

4- Ідеологія об'єктно-орієнтованого програмування

6.1. Теорія і **методи** структурного програмування

Методологія структурного програмування - це сукупність методів проектування та написання програм за жорсткими правилами, дотримання яких підвищує продуктивність праці програмістів, поліпшує читабельність і полегшує процес тестування програм [11]. Витоки цієї методології сягають 60-х років XX століття, коли швидкими темпами почала зростати складність програм, а отже, постала потреба у методах подолання такої складності. Методологія структурного -програмування ґрунтується на трьох методах: низхідного проектування, модульного програмування і структурування програм.

6.1.1. Низхідне проектування програм

В основу методу *низхідного проектування* покладено *алгоритмічну декомпозицію,* згідно з якою велика задача поділяється на дрібніші підзадачі, що їх можна розв'язувати окремо. Алгоритми розв'язання підзадач розглядаються як цілісні алгоритмічні блоки, або підпрограми, іменами яких можна оперувати при розв'язанні загальної задачі.

Отже, програма, що розв'язує загальну задачу, спочатку розглядається як незалежний модуль. Згодом вона поділяється на підпрограми, які декомпонуються на підмодулі наступного рівня. Процес декомпозиції триває доти, доки не будуть отримані блоки, що є достатньо малими для їх безпосереднього кодування. При цьому керуючу програму проектують раніше, ніж реалізують її складові частини.

Таким чином, програма ієрархічно структурується і розробляється шляхом послідовного уточнення на кожному рівні ієрархії. В основу цього процесу, крім принципу ієрархічності, покладено принципи абстрагування, специфікації інтерфейсів і модульності.

Абстрагування - це спрощений опис системи, в якому зосереджують увагу на певних властивостях і деталях, а на інші не зважають. Вдалою є та абстракція, що підкреслює суттєві деталі і відкидає несуттєві [8]. Під час низхідного проектування програми на верхніх рівнях абстракції деталі реалізації приховуються, а на нижніх рівнях вони описуються конкретною мовою програмування.

Специфікація інтерфейсів — це формалізований опис входів, виходів і функцій, що мають бути реалізовані програмним модулем. Коли інтерфейс модуля специфіковано, можна спробувати в явному вигляді записати його код. Якщо це зробити неможливо, модуль поділяють на певну кількість невеликих підмодулів. Коли модуль не можна ані закодувати, ані декомпонувати, його вважають *модулем-заглушкою,* інтерфейс якого відомий, а реалізація — ні.

Одним із прийомів формалізованого підходу до низхідного проектування є *метод ієрархічних діаграм,* що позначається абревіатурою HIPO (від англ. Hierarcьical Iприі Prоce55Iп§ Оиіриі - діаграма входу, обробки, виходу). Згідно з цим методом структура всієї програми подається у вигляді дерева, в якому підпрограми зображуються вузлами, а їхні виклики - ребрами.

6.1.2. Модульне програмування

Модульне програмування - це організація програми у вигляді сукупності незалежних блоків, структура і функції яких підпорядковуються певним вимогам. Такі блоки називаються модулями. Модульне програмування дає можливість:

- спростити процес тестування і налагодження програми;
- розробити програму зусиллями більше ніж одного програміста;
- локалізувати дію змін і доповнень до програми в межах одного модуля;
- скоротити термін і вартість розробки програми.

Програму можна вважати модульною, якщо її логічна і фізична структура відповідає таким умовам:

- модулі є незалежними програмними блоками, код яких логічно і фізично відокремлений від коду інших модулів;
- модулі є неподільними блоками програми, які можна використовувати повторно;
- розмір модуля обмежується розміром сегмента коду, який виділяється під час компіляції модуля.

Незалежність модулів є визначальним принципом модульного програмування. Це означає, що програма має бути поділена на модулі таким чином, щоб зв'язки між модулями були слабкими, а зв'язки всередині модулів - сильними. Точніше, модулі можуть вважатися незалежними, якщо вони задовольняють таким вимогам:

- кожен модуль можна замінити іншим функціонально еквівалентним модулем, що має той самий інтерфейс;
- взаємозв'язки між модулями встановлені за ієрархічним принципом;

- усі структури даних інкапсульовані в модулі, тобто доступ до даних може здійснюватися лише через процедури та функції модуля;
- модуль має одну точку входу та одну точку виходу;
- модуль повертає керування тому програмному блоку, що його викликав.

Під час поділу програми на модулі потрібно враховувати динаміку викликів процедур і функцій, а також їх розташування в оперативній пам'яті. Якщо поділ програми на модулі здійснено не раціонально, зростає кількість взаємних викликів підпрограм, розташованих у різних сегментах пам'яті, і це призводить до зниження швидкодії програми.

У разі колективної розробки модульної програми робота розподіляється так: більш досвідчені програмісти відповідають за інтерфейс модулів, а менш досвідчені — за їх реалізацію.

6.1.3. Методи структурування програм

Методи низхідного проектування та модульного програмування регламентують процес побудови архітектури програм на макрорівні. Методи структурування, навпаки, працюють на мікрорівні, регламентуючи процес створення програмного коду. В основу методів структурування програм покладено *теорему про структурування*, що її у 1965 році довели Бом і Джакопіні. Згідно з цією теоремою будь-яку програму можна побудувати з використанням лише трьох керуючих структур: послідовності, вибору і повторення.

Метою структурування є перетворення неструктурованої програми на еквівалентну їй структуровану, тобто таку, що складається з обмеженого набору керуючих алгоритмічних структур. Методи структурування грунтуються на поняттях функціонального вузла, а також на поняттях простої, елементарної і складеної програми [14].

Функціональніш вузлом називається частина алгоритму, що має один вхід та один вихід. Прикладом функціонального вузла є окремий оператор - присвоєння, виклик процедури тощо.

Програма називається *простою*, якщо вона має один вхід, один вихід і через кожен її функціональний вузол проходить деякий шлях від входу до виходу. Проста програма може містити прості підпрограми. Вся проста програма може розглядатись як один функціональний вузол.

Елементарна програма - це проста програма, що не містить підпрограм, які складаються більше ніж з одного вузла. Зазначимо, що існує лише обмежена кількість елементарних програм, з яких основними є три програми, які зображені на рис. 6.1.

Якщо функціональний вузол елементарної програми замінити елементарною програмою, утвориться *складена програма.* Якщо вузол складеної програми замінити елементарною програмою, знов утвориться складена програма. В такий спосіб із будь-якої множини елементарних програм утворюється певний клас складених програм. Так, множина {послідовність, іТ...іііеп...еІ5е} формує клас про-

грам без циклів, а множина {іт...іііеп...ei5e, міліє...сіо} приводить до утворення класу програм, які містять цикли. Отже, будь-яка підмножина множини елементарних програм є *базисною множиною* для певного класу складених програм. Складена програма, побудована з певної множини елементарних програм, називається *структурованою.*

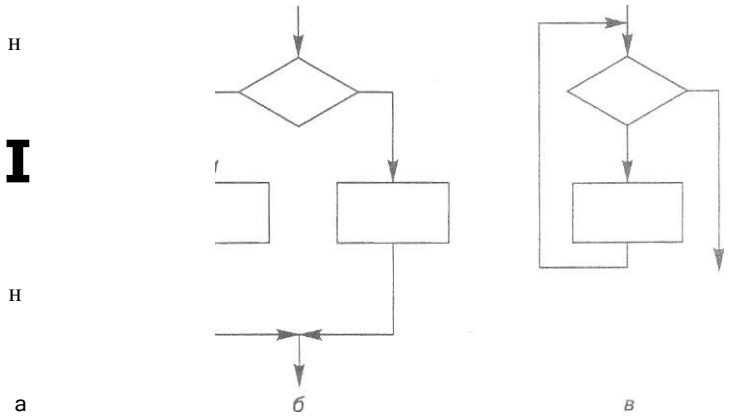


Рис. 6.1. Основні елементарні програми: послідовність (а), вибір (б), цикл із передумовою (в)

Позначимо літерою 5 клас складених програм, базисна множина якого містить такі елементарні програми, як послідовність (рис. 6.1, а), вибір (іТ...іііеп...eI5e, рис. 6.1, б) і цикл із передумовою (міїіе...сіо, рис. 6.1, в). Теорема про структурування стверджує, що будь-яку просту програму шляхом покрокового перетворення можна замінити функціонально еквівалентною структурованою програмою, що належить означеному вище класу 5. Згадане покрокове перетворення здійснюється за переліченими нижче правилами.

1. Правило *простоти:* створення програми слід починати із простої програми;
2. Правило *пакетування',* кожну дію можна замінити двома послідовними діями;
3. Правило *вкладання* - кожну дію можна замінити будь-якою структурою керування;
4. Правила 2 і 3 можна застосовувати у будь-якій послідовності необмежену кількість разів.

Принцип застосування правил пакетування та вкладання до простої програми продемонстровано на рис. 6.2.

Для перетворення неструктурованих програм у структуровані використовуються такі методи: дублювання кодів, введення змінної стану і метод булевих ознак [11].

Метод дублювання кодів застосовується для перетворення неструктурованих програм, що не містять циклів. Для отримання структурованої програми дублюють ті модулі, у які можна увійти з декількох точок.

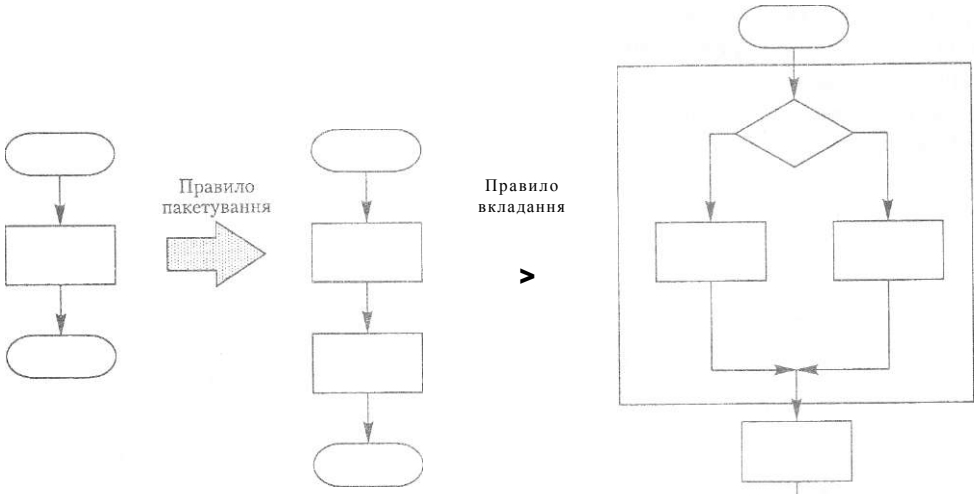


Рис. 6.2. Застосування правил пакетування та вкладання до простої програми

Приклад 6.1

Розглянемо неструктуровану програму, блок-схема якої зображена на рис. 6.3. Блоки позначені ідентифікаторами модулів.

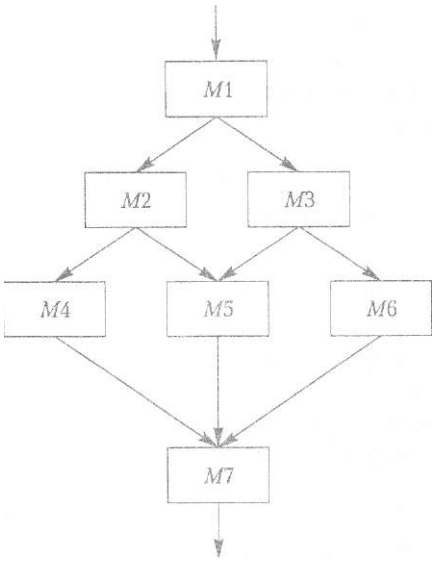


Рис. 6.3. Блок-схема неструктурованої програми

Із блок-схеми видно, що модуль М5 не може розпочати своєї роботи доти, доки не стануть відомими результати роботи модулів М2 і М3. Як правило, в модулі *М5* міститься фрагмент коду, що активізується під час виклику цього модуля з модуля М2, а також інший фрагмент коду, що активізується під час виклику модуля М5 з модуля М3.

Перетворимо зображену на рис. 6.3 неструктуровану програму, застосовуючи правила пакетування та вкладання. Зобразимо початкову програму як просту, що складається з елементарних програм типу вибору (іТ...їИеп...еІ5е). Користуючись правилом вкладання, розширимо блоки *МХ* і *МУ,* замінивши їх елементарними програмами типу іі...іііеп...еІ5е. У результаті заміни модуль М5 продублюється (рис. 6.4).

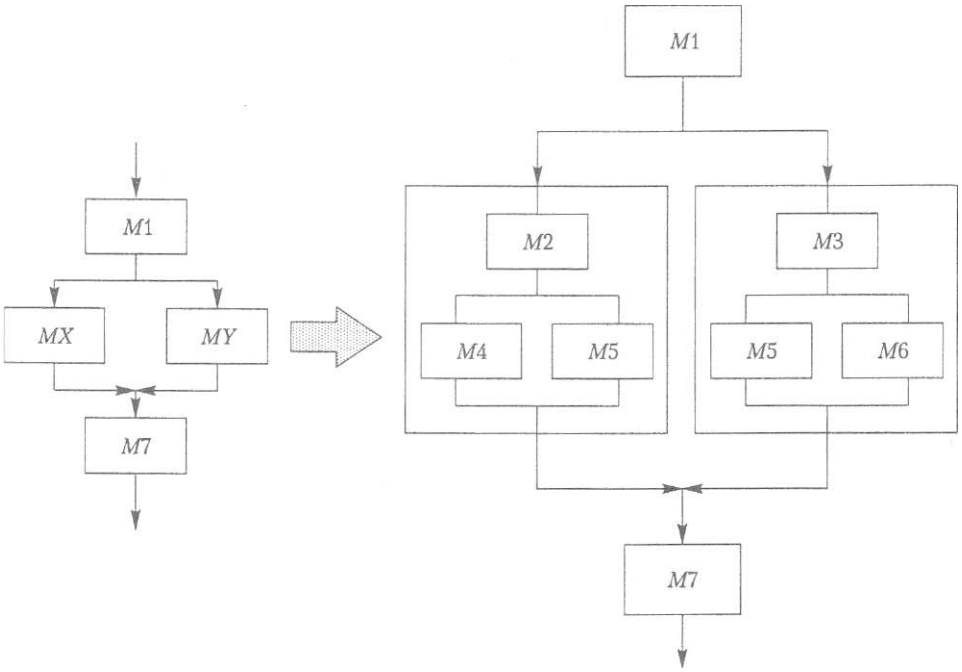


Рис. 6.4. Процес перетворення неструктурованої програми у структуровану

Метод введення змінної стану [11] застосовується до неструктурованих програм, що містять цикли. Процес перетворення неструктурованої програми у структуровану складається з таких кроків:

- у програму вводиться змінна стану, що її значення дорівнює номеру блока, на який буде передане керування після виконання поточних дій;
- функціональні блоки неструктурованої програми замінюються функціональними блоками, що виконують такі самі дії та надають змінній стану значення, що ідентифікує номер блока, на який передається управління.

Приклад 6.2

Розглянемо блок-схему реструктурованої програми, що містить цикли (рис. **6.5).** Кожному блоку приписано довільний номер.

У програму вводиться нова змінна *i.* Цій змінній присвоюється номер блока, на який потрібно передати керування. Зокрема, якщо істинна умова *A,* то цій змінній присвоюється номер блока *B,* тобто 2, якщо істинна умова *C,* змінній *i* присвоюється 1.... номер блока *A.*

Логічні блоки неструктурованої блок-схеми замінюються елементарними програмами типу IT...ИЄП...ЄІ5Є, що послідовно перевіряють значення змінної стану. Гілки «Так» програм типу іт.,.і1іеп...еі5е мають містити оператори присвоєння нових значень змінній стану і оператори перевірки певних умов початкової програми.

Блок-схему неструктурованої програми зображено на рис. 6.5, а блок-схему відповідної структурованої програми — на рис. 6.6.

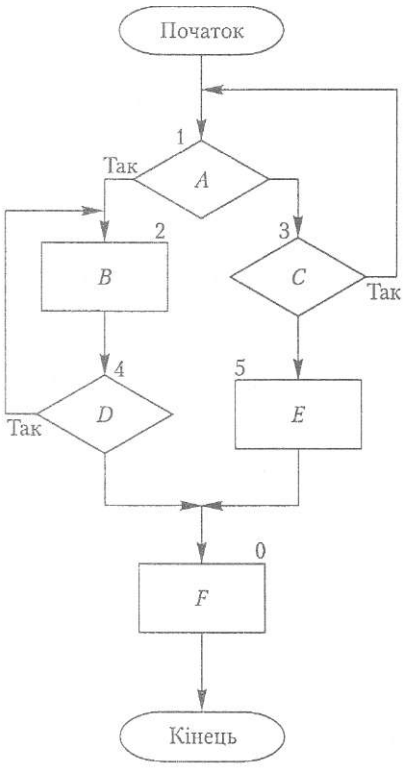


Рис. 6.5. Блок-схема неструктурованої програми що містить цикли

Метод булевої ознаки використовується для перетворення неструктурованих програм, що містять цикли. В програму вводиться додаткова змінна, яка набуває значення їгие чи їаїзє. Доки змінна ознаки зберігає це значення, виконання циклу триває. Значення змінної ознаки всередині циклу модифікується лише за певних умов.

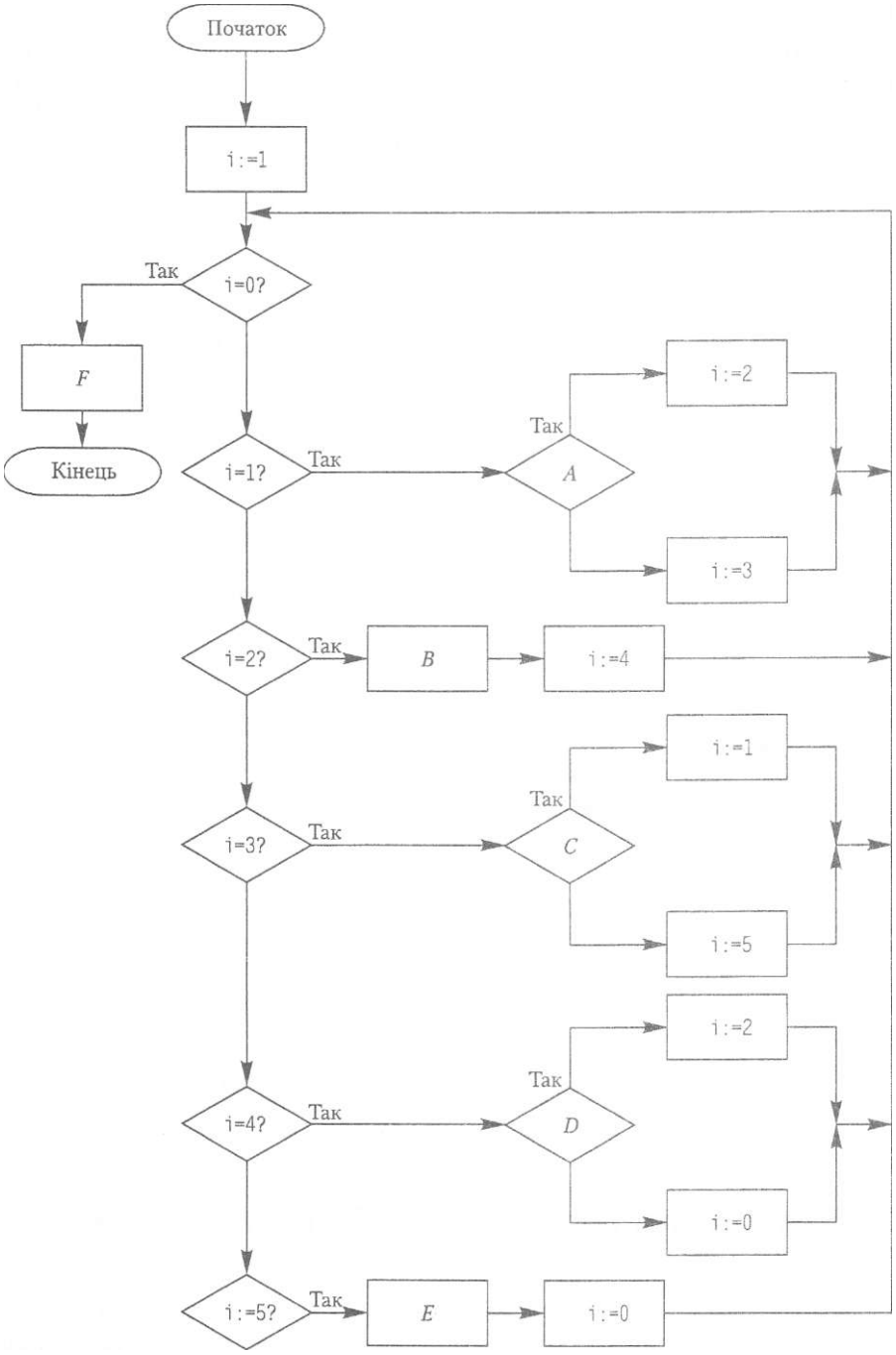


Рис. 6.6. **Блок-схема структурованої циклічної програми**

Приклад 6.3_

Розглянемо блок-схему неструктурованого циклу, що має одну точку входу та дві точки виходу (рис. **6.7). У** програму вводиться змінна **Лад,** що ініціалізується нулем **(Таїзе).** Якщо істинною є умова *P* або (}, змінна **П ад** набуває значення **1 (ігие),** що приводить до виходу з циклу. В такий спосіб неструктурований цикл із двома виходами перетворюється на структурований цикл з одним виходом (рис. 6.8).

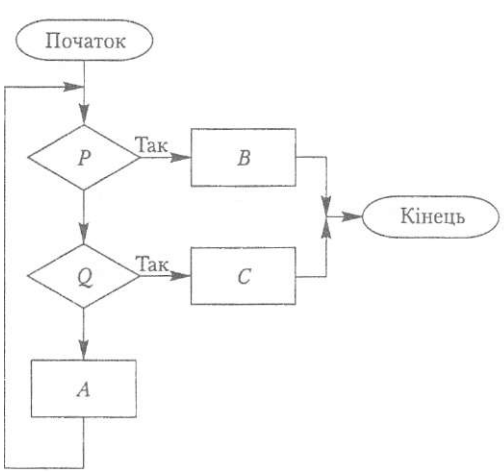


Рис. 6.7. **Блок-схема неструктурованого циклу**

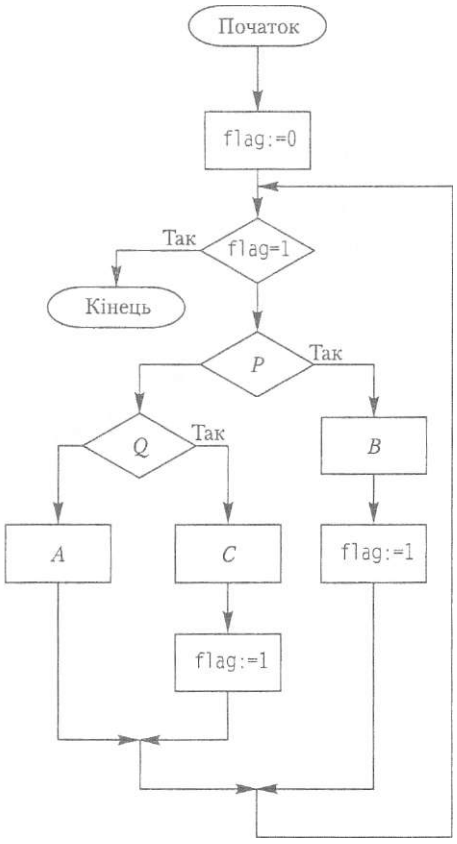


Рис. 6.8. **Блок-схема структурованого циклу**

6.2. Використання модулів у Вогіапсі РазсаІ 7.0

Модуль у мові РазсаІ — це програмна одиниця, що автономно компілюється в окремий бінарний файл ***.1ри** або ***.їруу.** До складу модуля можна включати оголошення констант, типів, змінних, а також оголошення і реалізацію процедур і функцій. Модуль з'єднується з різними Разсаі-програмами, що дає можливість використовувати у програмах ідентифікатори, оголошені в цьому модулі.

6.2.1. Структура модуля

Модуль складається із заголовка, інтерфейсної, реалізаційної й ініціалізаційної частин. Для оголошення модуля і перших двох його частин використовуються зарезервовані слова **ипі I, inlerTace, ітріетепїаііоп,** ініціалізаційна частина обмежується словами Ьеді п та **епсі** (з крапкою). Структуру модуля наведено нижче.

```
ипіїї          { — — — заголовок модуля »-"-      =
{*}            {директиви компілятора             -
іпїегґасе      {—        1ніерфейсна частина
изез           {розділ підключення модулів
солеї:         {розділ оголошення глобальних констант
їуре           {розділ оголошення глобальних типів
уаг            {розділ оголошення глобальних змінних
ргосесіиге     {розділ оголошення глобальних процедур
Гипсїіоп       {розділ оголошення глобальних функцій
ітр1етепїа£іоп {        реалізаційна частина      —
изез           {розділ підключення модулів
C0N5I          {розділ оголошення локальних констант
Iуре           {розділ оголошення локальних типів
уаг            {розділ оголошення локальних змінних
ргосесіиге     {реалізація процедур
Гипсїіоп       {реалізація функцій               —
Ьеділ          {—•-=== частина ініціалізації
епгі.
```

Заголовок модуля складається із зарезервованого слова ипі і та імені модуля:

```
ипі* <ім'я модуля>;
```

Ім'я модуля має збігатися з іменем бінарного файла цього модуля. Нагадаємо, що бінарний файл модуля має розширення Ірu або і створюється під час компіляції модуля. Сам модуль записується у файлі з розширенням раз.

Інтерфейсна частина

Інтерфейсна частина модуля починається із зарезервованого слова і **пег'Tace.** В цій частині містяться оголошення глобальних констант, типів, змінних, процедур і функцій, доступних програмам та іншим модулям, що використовують даний модуль. В оголошеннях процедур і функцій присутні лише їх заголовки. Тіла процедур і функцій записуються в реалізаційній частині модуля. Використання специфікатора "TomarcI у заголовках процедур і функцій модуля не допускається.

Приклад 6.4_

Розглянемо програму **ex4_6** з прикладу 4.6. Ця програма табулює значення декількох функцій на заданому відрізку. Оголошення типів, змінних і підпрограм запишемо в інтерфейсній частині модуля.

```
ипіїї ïaЬuIar;          {заголовок модуля          }
{========== інтерфейсна частина = = = = » — = = = = = }
inieri'ace
изез crі;
```

```
їуре Типс=Типсlіоп(х:геа1);геа1;          {процедурний тип}
уаг 1омег,иррег,5"Ьер:геа1;              {межі відрізка, крок}

Типсііоп Ро1упош(х:геа1):геаі; Таг;
Типсііоп 5іриз(х;геа1):геа1; Таг;
ргосесіиге РгіпШ.Ь.Іііreal;Т:Типе:5:зїгіпд);
```

Реалізаційна частина

Реалізаційна частина модуля починається із зарезервованого слова **ішрієшепїаіїоп** і містить тіла процедур і функцій, оголошених в інтерфейсній частині. Реалізаційна частина може містити також оголошення констант, типів, змінних, процедур і функцій, що є локальними в межах модуля, тобто доступними тільки для процедур і функцій цього модуля, але недоступними для інших модулів програми, в якій використовується даний модуль. Усі підпрограми, заголовки яких наведено в інтерфейсній частині модуля, повинні бути реалізовані в його реалізаційній частині.

Приклад 6.5_

Розглянемо реалізаційну частина модуля, інтерфейсну частину якого наведено у прикладі 6.4. Зауважимо, що у разі, коли модуль не має частини ініціалізації, за реалізаційною частиною записується операторна дужка епсі. (з крапкою).

```
ипіі Іаьиіаг;              {заголовок модуля              }
{========== інтерфейсна частина ================}
іпІегТасе
{наведено у прикладі 5.4                          }
{-=-—»=== частина реалізації модуля ============}
ітрієтепїаііоп
  Типсїіоп РоїупотСх:геаі):геаі;
  Ьеділ
    Ро1упот:=5цг(х)*х-х+1;
  епсі:

  Типсїіоп 5іриз(х:геа1):геаі;
  Ьеділ
    5іриз:=зіп(х);
  епсі;
  {а. Ь - межі відрізку, на якому табулюється функція;
       Ь - крок зміни аргументу;
       Т - функція, що табулюється;
       з - назва функції у текстовому вигляді}
  ргосесіиге РппІ(а,Ь,Ь:геа1:"Г:Типе;5;5Їгіпд);
  уаг х:геаі;
  Ьеділ
    ^П "Ьеі П ('==========');
    мгі1:е1п('|  х   [   ',з);
    МГНЄІП('==========');
    х:=а;
    мітіїе хЧь сіо
```

```
    Ьеgіп
      мгі£е(x:6,'|');
      мп£е(Т(x):10:6);
      мгіієіп;
      x:=x+H;
    епсі;
  епсі;
епсі.
```

Ініціалізаційна частина

Ініціалізаційна частина починається з оиераторної дужки **Ьеgіп** і завершується операторною дужкою епсі, після якої ставиться крапка. Ініціалізаційна частина модуля може містити певний фрагмент програми. Оператори з цього фрагмента виконуються до передачі керування програмі, у якій використовується модуль.

Приклад 6.6

```
иніі їаЬиіаг;          {заголовок модуля}
;========== інтерфейсна частина ==========}
iпїегTace
{наведено у прикладі 6.4                    }
{========== частина реалізації модуля ========== }
ішріетепїаїіоп
{наведено у прикладі 6.5                    }
{======== частина ініціалізації модуля ========== }
Ьеgіп
  мгііе1п('Початок програми');
епсі.                  {кінець модуля}
```

6.2.2. Компіляція І використання модулів

Розроблений на мові PascaI модуль зберігається як раз-файл. У результаті його компіляції за допомогою команди сотріїе меню Сотріїе створюється бінарний файл із розширенням фи (від англ. ТигЬо PascaI Пдії). Під час створення виконуваного файла програми, яка використовує модулі, або під час її запуску Трп-файли з'єднуються з її відкомпільованим кодом. В ГОЕ BorlancI PascaI 7.0 таке з'єднання виконується командами таке і Ьиіїсі меню Сотріїе.

Модуль підключається до програми за допомогою директиви изез, що має бути розташована на початку програми:

изез **<ім'я модуля>;**

У разі використання команди таке компілятор перевіряє наявність їри-файлів для кожного з оголошених в операторі **изез** модулів. Якщо вхідний файл змінився, його буде перекомпільовано. Якщо в інтерфейсну частину модуля внесені зміни, то перекомпільовуються всі модулі, що звертаються до нього. Під час виконання команди ЬиіїсІ перекомпілюються всі модулі, що використовуються у програмі і згадані в операторі изез. У прикладі 6.7 показано використання модуля їаЬиіаг, код якого наведено у прикладах 6.4-6.6

Приклад 6.7

Запишемо програму., що тричі табулює многочлен на різних відрізках і з різним кроком. Зауважимо, що під час компіляції програми модуль **(3bviar** має бути досяжним для компілятора, тобто шлях до нього необхідно вказати у полі **Ilnii Oirec-**їогіез вікна, що відкривається за допомогою команди **Opiionz • Pireciorie3.**

```
рrодгаш ехб_1;
изез РаЬиIаг.crI;
маг
  i: i пi;едег:
Ьедiп
  Tor i:=1 io 3 cio
  Ьедiп
    мпїеiп('епiег 1омег.иррег Ьоипсіз апсі зіер'):
    reaciiп(Iомег. иррег, зіер);
    Pгiпi(1 омег,иррег,зїер,Роїупош):
  епсі;.
епсі,
```

6»3, Основні концепції об'єктно-ор.ієнтованої методології програмування

Методологія об'єктно-орієнтованого програмування виникла як результат природної еволюції мов структурного програмування. З погляду цієї методології програма в сукупністю об'єктів, кожен об'єкт є екземпляром певного класу, а класи утворюють ієрархію успадкування [4].

©.3.1. Базові поняття об'єктно-орієнтованого програмування

Пояснимо поняття, що були використані у визначенні об'єктно-орієнтованої методології. *Об'єкт* — це реальна або абстрактна сутність, яка моделює частину навколишньої дійсності. Отже, кожний реальний предмет — це об'єкт. Прикладами реальних об'єктів є автомобіль, літак, завод, людина, матриця, вектор тощо. Ті самі слова: «автомобіль», «літак», «людина» тощо позначають не об'єкти, а *класи,* коли йдеться не про конкретний автомобіль, літак або людину, а, наприклад, про автомобіль або літак як різновид транспортного засобу і про людину як біологічний вид. Отже, клас є абстракцією множини об'єктів, що мають спільні властивості і поведінку. Транспортний засіб і біологічний вид — це теж класи. Відношення між літаком і транспортним засобом або між видом «Ьото заріепз» і біологічним видом взагалі є відношенням *успадкування,* або відношенням «є» (англ. «із а»): літак є різновидом транспортного засобу, вид «Ьото заріепз» є різновидом біологічного виду. Коли клас *В* є різновидом класу *А,* то *А* називається класом-предком, *В* — класом-нащадком.

З погляду програмування об'єкт складається з атрибутів і методів. *Атрибути* описують властивості об'єкта у певний момент часу, *методи* — властиву для об'єк-

та поведінку. Всі об'єкти, що є екземплярами одного класу, мають однаковий набір атрибутів і методів. Значення атрибутів зберігаються в змінних, а дії методів описуються в процедурах або функціях Тому клас може бути визначений як набір оголошень змінних-атрибутів і підпрограм-методів. Визначені всередині класу елементи даних називаються *змінними-членами* класу, процедури і функції - *функціями-членами,* або методами класу. Оголошення класу називається його *інтерфейсом,* а опис його методів — *реалізацією.* Як правило, інтерфейс класу відокремлюється від його реалізації.

Об'єктно-орієнтоване програмування грунтується на трьох концепціях: інкапсуляції, успадкування і поліморфізму. *Інкапсуляція* — це механізм, який дозволяє захистити атрибути й методи об'єкта від некоректного використання. Згідно з принципами інкапсуляції атрибути класу не можуть бути доступними для екземплярів інших класів безпосередньо. Доступ до атрибутів має здійснюватися лише через методи класу. Наприклад, доступ до двигуна автомобіля можна здійснити лише за допомогою методів «завести», «вимкнути», «перемкнути швидкість» тощо. Саме завдяки інкапсуляції можна отримати зиск у разі відокремлення інтерфейсу класу від його реалізації. Адже інкапсуляція дає можливість зробити програми, що використовують об'єкти певного класу, незалежними від способу реалізації цього класу. •

Успадкування дозволяє систематизувати подібні класи на підставі їх спільних властивостей. Клас, що містить атрибути і методи, спільні для групи подібних один до одного класів, називається *базовим класом,* або класом-предком. Класи, що успадковують властивості і функціональні особливості базового класу, називаються *похідними класами,* або класами-нащадками. Як уже зазначалося, успадкування застосовують для класів, пов'язаних семантичним відношенням «є». Іншими словами, вважається, що екземпляр похідного класу водночас є екземпляром базового класу. Це дає можливість об'єктам-нащадкам використовувати атрибути і методи об'єктів-предків як свої власні (хоча слід зазначити, що деякі мови програмування, наприклад C++, дозволяють встановлювати обмеження на доступ до атрибутів і методів предків з боку методів нащадків). Найважливішою властивістю успадкування є те, що воно дає можливість уникнути повторень коду, адже спільний для множини подібних класів код може бути винесений в методи їх спільного предка. За допомогою механізму успадкування можна побудувати ієрархію класів.

В об'єктно-орієнтованому програмуванні взаємодія між об'єктами здійснюється шляхом передачі повідомлень. Об'єкт, що отримав повідомлення, реагує на нього викликом відповідного методу. Оскільки кожен об'єкт-нащадок є водночас будь-яким своїм предком, то надіслане такому об'єкту повідомлення надсилається насправді декільком об'єктам різних класів. Рішення про те, якому саме об'єкту слід опрацювати повідомлення, залежить від обставин, що з'ясовуються під час виконання програми. Здатність об'єктів похідних класів по-різному реагувати на ті самі повідомлення, називається *поліморфізмом.* Поліморфізм дає можливість визначати метод, що його буде викликано, під час виконання програми, а не під час її компіляції, а також можливість використовувати однойменні методи з різними заголовками.

6.3.2, Класи і об'єкти в мові Pascal

Наведемо синтаксис оголошення класу в мові PascaI. Зазначимо, що в цій мові клас називається об'єктним типом даних і позначається ключовим словом oi^ecі.

```
їуре <ім'я об'єктного типу> = oЬзecI;
          <оголошення атрибутів>;
          <заголовки методів>;
        епсі;
```

Тут **<ім'я об'єктного типу>** - довільний ідентифікатор; **oi**^ecі — слово, зарезервоване для позначення об'єктного типу даних; **сотолошення атрибуті в>** — перелік оголошень змінних будь-яких припустимих типів; **сзаголовки методів>** — заголовки будь-яких процедур або функцій.

Після того як клас оголошено, його можна використовувати для створення ' екземплярів класу, тобто статичних або динамічних змінних об'єктного типу. Зрозуміло, що оголошення класу має передувати оголошенню і використанню екземпляра цього класу. Наведемо синтаксис оголошення екземплярів об'єктного типу в мові PascaI:

```
маг сім'я змінно!'-екземпляра>: сім'я об'єктного типу>;
```

Під час оголошення методу використовується складене ім'я, що містить розділені крапкою імена об'єктного типу і методу. Оголошення методів об'єктів має відповідати такому синтаксису:

```
ргосесііиге сім'я об'єктного типу>.сім'я процедури(параметри)>:
Ьеділ
  {тіло процедури}
епсі;
ГипсЬіоп сім'я об'єктного типу>.сім'я функці1(параметри)>:<тип>;
Ьеділ
  {тіло функції}
епсі;
```

Екземпляри об'єктного типу можна використовувати для виклику методів об'єкта. Виклик методів об'єкта здійснюється з використанням складеного імені, що містить ім'я екземпляра об'єктного типу та ім'я методу:

```
сім'я ЗМІННОІ-ЄКЗЄМПЛЯра>.сім'я мєтоду>(ссписок аргументі 8>)
```

Так само здійснюється доступ до атрибутів об'єктів певного класу з методів іншого класу:

```
сім'я змінної-екземпляра>.сім'я атрибуту>
```

Отже, призначення методів - реалізація дій над елементами даних об'єктного типу. Методи об'єктного типу мають прямий доступ до атрибутів цього типу, оскільки вони містяться в одній області видимості. Тому, коли метод звертається до атрибутів того об'єктного типу, якому належить він сам, складені імена не використовуються.

Приклад 6.8

Розробимо об'єкт «вектор», атрибутами якого є одновимірний масив і його розмірність (масиви розглядатимуться в розділі 7). Опишемо такі методи об'єктного типу, як ініціалізація та відображення вектора, а також обчислення скалярного добутку двох векторів.

```
Упіі IInii:6_2;                    {модуль-(файл ипїі6_2.раз)}
іпїегТасе                          {розділ інтерфейсу        }
изез сгї;
їуре
  шаз=array[1.,10]оТ inРедег;
  уес£or=оЬоесі:                   {об'єктний тип УЄСЇОГ     }
     п:іпіедег;                    {розмірність вектора      }
     уес;таз;                      {значення елементів вектора}
     пате :сИаг;                   {ім'я вектора             }
     procesciиre іпіі(сИ:сИаг);{ініціалізація даних      }
     procesciиre зliомуес(cli:cliar); {відображення значень
                                       елементів             }
     procesciиre зсаіаг(V:УЄСІОГ) ; {скалярне множення
                                                  векторів}
  епсі;
ітріетепіаїіоп                     {розділ реалізації методів }
{========== метод ініціалізації вектора ==========}
procesciиre Уесїог.іпі£(сЬ:сЬаг):
  уаг і :іпі;едег;
  Ьеділ
    пате:=сИ;
    мгїїе('іприи; їИе пишьег оТ УЄСІОГ ',
            пате,' сотропепїз;  ');
    reaciin(п);
    мгїieInCiприи: '.п.'  сотропепіз ');
    Тог і :=1 -Ьо п сіо
        reac!(уес[і]);
  епсі;
{========== метод виведення вектора ====————===}
procesciиre уесlor.зliомуес(сН:cliar);
уаг і :іп1едег;
Ьеділ
  пате;=сЬ;
  мгііе1п('УЄСІОГ  '.пате,';');
  Тог і :=1 їо п сіо  мгі1;е(уес[і]. '  ');
  мгїїеіп;
епсі;
{==== метод обчислення скалярного добутку векторів ====}
procesciиre уесі;ог.зса1ar(у:уесі;ог);
                                   {V - екземпляр об'єктного типу}
уаг $ит.                           {скалярний добуток векторів  }
     і;inРедег;                    {параметр циклу              }
Ьеділ
іТ поу.п -ЬИеп Ьеділ   {коли розмірності векторів різні}
  мгїieInC'зсаіаг ргосисР із пої сошриіаЬІе;  ');
  мгіle1п('УЄСЇОГ  IепдіНз are сііТТегепї'):
```

```
  reacПп;
  exii;
epci
eize                    {коли розмірності векторів однакові}
Ьедіп
  зит:=0;
  Tor i :=1 io n cio          {скалярний добуток векторів}
    зиш:=зит+yec[i]*У.УЄС[I];
  мгііе!п('зcaiar rrocисї = ',зиш);
epci:
epci;
epci.                                   {кінець модуля}

rrодгаш ex6_2;      {програма роботи з об'єктом «вектор»}
изез cr'Ъ. иіпГЪб_2:
yar У1,У2:УЄСІОГ:                  {об'єкти-вектори    }
Ьедіп
  cirзcr;
  мгile1n('creaie УЄСЇОГЗ
  У1.inП('А');          {ініціалізація вектора А    }
  У2.ІПії('В');         {ініціалізація вектора В    }
  У1.ЗШОМУЄС( 'А');     {виведення вектора А        }
  У2.ЗНОМУЄС('В1);      {виведення вектора В        }
  у1.зса1ar(y2);{обчислення скалярного добутку векторів}
epci.
```

```
• (IпасПre C:\Ш>\ПпіХ6 ?ГХП                    Pixi
creaie иесіогз
іприї 1(ie питЬег o? иесїог й сотропепїї:  З
іприї: 3 сотропепіз
1 2 3
іприї: ЇГіе питЬег o? иесіог В сотропепіз:  З
іприї 3 сопропепіз
-3 -2 -1
чесіог Я:
1 2 3
ліесіог В:
-3 -2 -1
їcaiar rровисі = -10
```

РИС. 6.9. Результати роботи програми ex6_2. Обчислення скалярного добутку векторів

Висновки

- Методологія структурного програмування — це сукупність методів проектування та написання програм за жорсткими правилами, дотримання яких підвищує продуктивність праці програмістів, поліпшує читабельність програм і полегшує процес їх тестування. Методологія структурного програмування ґрунтується на трьох методах: низхідного проектування, модульного програмування і структурування програм.

- В основу методу низхідного проектування покладено алгоритмічну декомпозицію, згідно з якою велика задача поділяється на дрібніші підзадачі, кожну з яких можна розв'язувати окремо. Такий процес ґрунтується на принципах ієрархічності, абстрагування, а також специфікації інтерфейсів і модульності.
- Абстрагування - це спрощений опис системи, в якому зосереджують увагу на певних властивостях і деталях, а на інші не зважають.
- Специфікація інтерфейсів — це формалізований опис входів, виходів і функцій, що мають бути реалізовані програмним модулем.
- Модуль має бути незалежним блоком програми, код якого логічно і фізично відділений від коду інших модулів, завдяки чому модуль можна модифікувати без наслідків для інших модулів і повторно використовувати.
- В основу методів структурного програмування покладено теорему про структурування, згідно з якою будь-яку програму можна побудувати з використанням лише трьох керуючих структур: послідовності, вибору і повторення.
- Розробляючи структуровані програми, слід дотримуватися таких правил: правила простоти (створення програми починати з простої програми), правила пакетування (кожну дію можна замінити двома послідовними діями), правила вкладення (кожну дію можна замінити будь-якою структурою керування).
- Для перетворення неструктурованих програм у структуровані використовуються методи дублювання кодів програми, введення змінної стану і булевих ознак.
- Модуль у мові РазсаІ — це програмна одиниця, що автономно компілюється в окремий бінарний файл *.1ри або *.1рм. До складу модуля можна включати оголошення констант, типів, змінних, а також оголошення і реалізацію процедур і функцій. Модуль з'єднується з різними Разсаі-програмами, що дає можливість використовувати у програмах ідентифікатори, оголошені в модулі.
- Модуль складається із заголовка, інтерфейсної, реалізаційної й ініціалізаційної частин. Для означення модуля та його частин використовуються зарезервовані слова **ипі 1; іпїегТасе, ішріешепіаіїоп.**
- Методологія об'єктно-орієнтованого програмування дозволяє розглядати програму як сукупність об'єктів, кожен з яких є екземпляром певного класу, а класи утворюють ієрархію успадкування.
- Об'єкт — це реальна або абстрактна сутність, що моделює частину навколишньої дійсності. З погляду програмування об'єкт складається з атрибутів і методів. Атрибути описують властивості об'єкта у певний момент часу, методи — властиву об'єкту поведінку.
- Клас є абстракцією множини об'єктів, що мають спільні властивості і поведінку.
- Об'єктно-орієнтоване програмування ґрунтується на концепціях інкапсуляції, успадкування і поліморфізму.
- Інкапсуляція - це механізм, який дозволяє захистити атрибути й методи об'єкта від некоректного використання.
- Успадкування дозволяє систематизувати подібні класи на підставі спільності їх властивостей. Клас, що містить атрибути і методи, спільні для групи подібних

один до одного класів, називається базовим класом, або класом-предком. Класи, що успадковують властивості і функціональні особливості базового класу, називаються похідними класами, або класами-нащадками. Найважливішою властивістю успадкування є те, що воно дає можливість уникнути повторень коду.

+ В об'єктно-орієнтованому програмуванні взаємодія між об'єктами здійснюється шляхом передачі повідомлень. Здатність об'єктів похідних класів порізному реагувати на ті самі повідомлення, називається поліморфізмом.

Контрольні запитання та завдання

1. У чому полягає метод низхідного проектування?
2. Що таке алгоритмічна декомпозиція?
3. Що таке програмний модуль?
4. Чим відрізняється проста програма від елементарної?
5. Сформулюйте теорему про структурування.
6. Викладіть сутність методів структурування циклічних програм.
7. Які оператори неприпустимо використовувати у структурованій програмі?
8. Яку структуру має модуль у мові РазсаІ?
9. Дайте визначення методології об'єктно-орієнтованого програмування.
10. Чим відрізняється клас від об'єкта?
11. Які концепції покладено в основу об'єктно-орієнтованого програмування?
12. Які вигоди можна отримати від використання інкапсуляції?

Вправи

1. Доповніть твердження.
   1.1. В мові РазсаІ для оголошення об'єктного типу використовується ключове слово_____.
   1.2. Для доступу до елементів даних і функцій-членів об'єктного типу використовується_____ім'я.
   1.3. У структурному програмуванні використовуються такі базові алгоритмічні структури:_____.
   1.4. Кожний блок структурованої програми має_____вхід та_____вихід.
2. Виберіть правильні визначення.
   2.1. Інкапсуляція - це абстрагування об'єктів.
   2.2 Об'єкт — це інкапсульована абстракція.
   2.3. Об'єкт — це екземпляр класу.
   2.4. Клас — це множина об'єктів.

3. Виберіть правильні твердження і обґрунтуйте їх:
   3.1. Похідний клас містить усі методи і атрибути свого базового класу.
   3.2. Базовий клас містить методи і атрибути всіх своїх похідних класів.
   3.3. Похідний клас може мати декілька базових класів.
   3.4. Базовий клас може мати декілька похідних класів.
   3.5. Об'єкт похідного класу можна розглядати як об'єкт базового класу.
   3.6. Об'єкт базового класу можна розглядати як об'єкт похідного класу.
4. На рис. 6.10 зображені блок-схеми неструктурованих програм. Побудуйте блок-схеми еквівалентних структурованих програм.

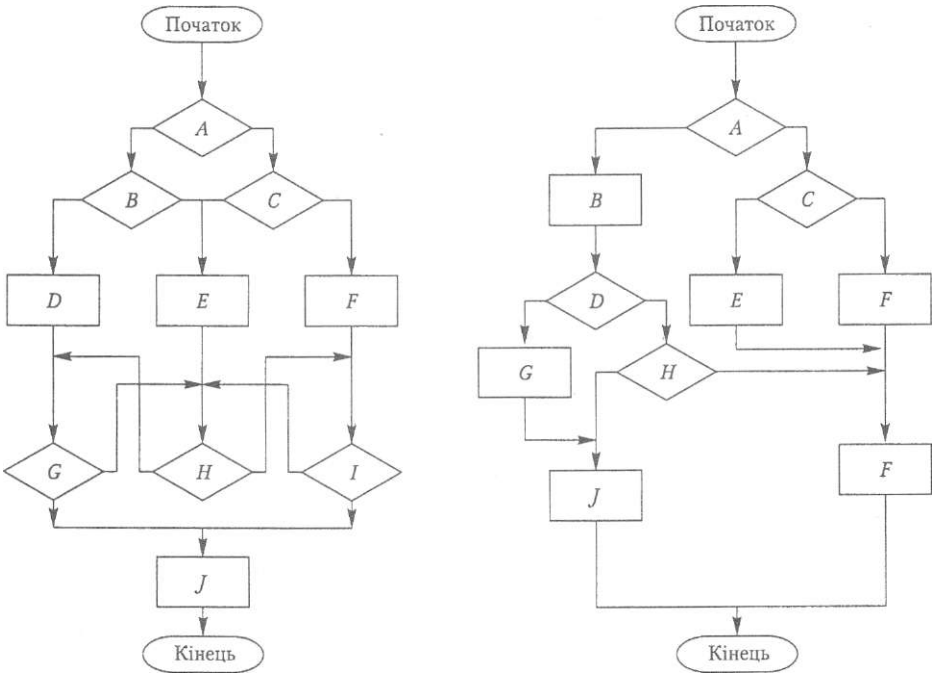


Рис. 6.10. Блок-схеми неструктурованих програм

5. Визначте результат роботи таких фрагментів коду, якщо x = 9, y = 11:

5.1. iT **x < 10** Ilieп iT **y > 10** TЬеп **мгі1;е1п('****')**
eізе **игіїеіп('####'); мгіїеlnC '$$$$');**

5.2. iT **x < 10** IHеп Ьеділ
IT **y > 10** їИеп **мгіі:е1п('****')**; епсі
eізе Ьеділ **мгіїе!п('####'); игіїеіп('**$$$$[1]): епсі;

6. Структуруйте наведену нижче програму.

```
Var x:in1едег;
Ьеділ
  Tor x:=1 £o 10 cio
  Ьеділ
```

```
    iT x = 5 1:ben conііпие;
    мгіі;e1n(x,'  '):
  епсі;
  мгїieInC'изе соііпие їо зкір x=5');
епсі.
```

7. Визначте, які з програм, зображених на рис. 6.11, є простими.

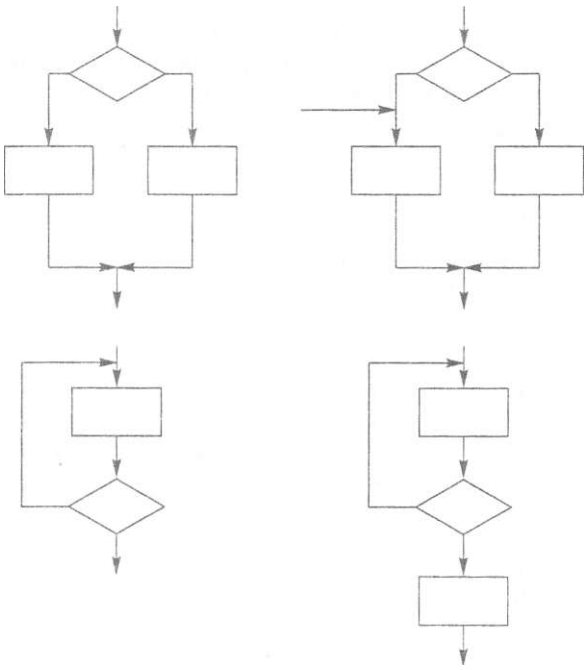


Рис. 6.11. **Блок-схеми програм різної структури**

8. Замініть неструктуровану програму, що використовує оператор безумовного переходу **доіо,** її структурованим еквівалентом. Якщо введено додатне значення змінної **x,** програма його збільшує, в противному разі операція введення повторюється.

```
IaЬeI 1.2;
yar x:in1едег;
Ьеділ
  2:мгіle1n('enier x');
  reaciInCx);
  іґ x > 0 IHen доіо 1
  еізе Ьеділ мгііеіп Ггереаі епїег x');доїо 2; епсі;
  1; Ьеділ
    x:=x+10;
    мгіі;e1n(x);
  епсі;
епсі.
```

Частина 2

Організація **даних**

Розділ 7

Масиви

4- Поняття одновимірного масиву та його властивості
- Типові операції над одновимірними масивами
- Поняття багатовимірного масиву

4 Типові операції над багатовимірними масивами

4 Рядковий тип даних

7.1. Одновимірні масиви

Характерною ознакою простих типів даних є те, що вони атомарні, тобто не містять як складові елементи дані інших типів. Типи даних, що не задовольняють зазначеній властивості, називаються структурованими. У мові РазсаІ означено такі структуровані типи: масиви, рядки, множини, записи та файли. Структурований тип характеризується множиною елементів, з яких складаються дані цього типу. Елементи структурованого типу називаються *компонентами* (компоненти масивів найчастіше називають *елементами масивів*). Отже, змінна або константа структурованого типу завжди містить декілька компонентів. У свою чергу, кожний компонент може належати до структурованого типу, що дозволяє говорити про вкладеність типів. З простих елементів даних, або компонентів, більш складні структури утворюються двома способами: об'єднанням однорідних елементів даних з метою створення масивів та об'єднанням різнорідних елементів даних з метою створення записів.

Потреба у масивах виникає тоді, коли в оперативній пам'яті зберігається велика, але визначена кількість однотипних даних. Наприклад, можна задати масив щоденних значень температури повітря протягом місяця з метою визначення середньої температури або масив логічних значень, що зображуватиме наявність квитків на кіносеанс на всі місця у кінозалі. Розрізняють одновимірні та багатовимірні масиви. Зокрема, масив температур є одновимірним, а масив квитків — двовимірним. Одновимірні масиви розглядаються в розділі 7.1, а багатовимірні - в розділі 7.2. У розділі 7.3 розглянуто рядки, які є різновидом одновимірних масивів.

7.1.1. Поняття масиву та його властивості

Дамо означення одновимірного масиву, розрізняючи тип масиву та дані цього типу. Терміном «масив» надалі позначатимемо саме дані деякого типу масиву.

Одновішірний масив - це послідовність однотипних даних.

Уважно проаналізувавши це означення, можна зробити висновок, що масив фактично поєднує в собі дві структури: множину елементів і заданий на цій множині порядок. Усі елементи масиву мають один і той самий тип, що називається *базовим.* З іншого боку, порядок теж визначається набором значень одного й того самого типу, що називається *індексним,* а самі ці значення називаються *індексами.* Кожному елементу масиву відповідає певний індекс. Індексний тип має бути простим порядковим типом даних. Кількість елементів в одновимірному масиві називається його *розмірністю,* або *довжиною.*

Тип масиву - це структурований тип даних, множина допустимих значень котрого складається з усіх масивів, для яких зафіксовано:

- розмірність;
- базовий тип;
- індексний тип;
- множину значень індексу.

Щойно наведене визначення типу масиву можна застосувати до типів як одновимірних, так і багатовимірних масивів. Зауважимо також, що усі елементи одновимірного масиву записуються до розташованих поряд ділянок оперативної пам'яті. Тому і весь масив може розглядатися як одна нерозривна область пам'яті.

З точки зору математики одновимірний масив - це вектор. Наприклад, масив або вектор *А,* що має п'ять елементів, які записують у математиці у вигляді індексованих змінних a_1, a_2, a_3, a_4, a_5, можна зобразити значеннями цих змінних у сусідніх ділянках оперативної пам'яті.

a_1	a_2	a_3		a_5

Напрям збільшення адрес пам'яті

Ідентифікатор типу масиву можна оголосити в розділі **iype** з використанням такого синтаксису:

<ім'я типу масиву> = array **[<нижній індекс>..<верхній індекс>]** of **<тип елементів>:**

У цьому оголошенні **array, oT** — зарезервовані слова, що перекладаються як «масив», «з»; **<ім'я типу масиву>** - деякий ідентифікатор; **<тип елементів>** - будь-який тип даних, окрім файлового типу; **<нижній індекс>** і **<верхній індекс>** — константи, що визначають межі діапазону допустимих значень індексу. Розмірність масиву дорівнює величині **orcI(<верхній індекс>)-orcI(<нижній індекс>)+1.**

У розділі уаг оголошується змінна, що матиме раніше оголошений тип масиву:

<ім'я масиву> : <ім**'я типу масиву>;**

Синтаксис мови РазсаІ дає можливість поєднати у розділі уаг оголошення змінної-масиву із визначенням її типу. При цьому ідентифікатор типу масиву не оголошується:

<ім'я масиву> : array **[<нижній індекс>..,<верхній індекс>]** oT **<тип елементів>:**

Нагадаємо, що обсяг пам'яті, яка виділена для зберігання всіх оголошених у розділах уаг змінних, не повинен перевищувати 64 Кбайт. Тому є обмеження на максимальну кількість елементів у масиві. Так, максимальна кількість елементів типу **іпіедег** не може перевищувати 32 767, а елементів типу **reai** — 10 922.

Оголосити змінну типу масиву можна і з використанням такого синтаксису:

<ім'я масиву> : аггау **[<тип індексів>]** оТ **<тип елементів>;**

Тут <тип індексів> — ЦЛІ ТИПИ **зюгіппі** або Ьуіе, для яких кількість допустимих значень становить 256, або оголошений в розділі їуре перелічуваний тип. Як типи індексів не дозволяється вказувати типи іпїедег, могсі і Іопдіпі, оскільки розмір оголошеного в такий спосіб масиву становив би не менш ніж 64 Кбайт.

Приклад 7.1_

Наведемо декілька прикладів оголошень одновимірних масивів.

```
Рrодгаш ex7_1;
соп5І:
  5їarP=10:                                          {нижній індекс}
  Tinizii=30;                                        {верхній індекс}
їуре
  ciay=(Monclay, Тиезciay. Иесіпезciay, Тіїигзciay, Prіciay,
        Zaiirciay. Zimciay);                         {перелічуваний тип}
  yec£or=аггау[1..10] оТ іпіедег;
  5TK=аггау[Ьуі:е] оТ cHar:
уаг
  aiyecїor;                                  {масив 10 змінних типу іпїедег}
  ЗІГ:3TK;                                {масив 256 змінних символьного типу}
  ciidii::аггау[5. .20] оТ іпїедег;      {масив 16 змінних типу іпїедег}
  T1oal:аггау[zi:ari:. .TinizЬ] oT reai:    {масив 21 змінної типу reai}
  Ietpегаїиге:аггау[clay] оТ reai;
Ьеdіп
мгі1;е1п( 'ex7_1'):
епсі.
```

Підсумовуючи матеріал розділу 7.1.1, назвемо основні властивості масивів, притаманні як одновимірним, так і багатовимірним масивам:

- однорідність — усі елементи належать одному типу;
- сталість — вимірність масиву задається під час його оголошення і не змінюється протягом роботи з ним;
- рівнодоступність — спосіб доступу до всіх елементів є однаковим;
- послідовність розташування — усі елементи масиву розташовані в послідовних комірках оперативної пам'яті;
- індексованість - елементи однозначно ідентифікуються своїми індексами;
- упорядкованість індексу - індексний тип має бути простим порядковим типом даних.

7.1.2. Базові операції обробки одновимірних масивів

Будь-яка обробка масивів здійснюється шляхом виконання операцій над їх елементами. Двома найпростішими операціями над елементами одновимірного масиву є вибір певного елемента та зміна його значення. Для доступу до окремого елемента масиву застосовується операція індексування [], за допомогою якої утворюються вирази **<1 м' я масиву>[<1 ндексний вираз>].** Елемент масиву є окремою змінною, що ідентифікується вище зазначеним виразом. Приклад ідентифікації елементів масиву а з індексами I, 2, ..., 10 та масиву **сИдіІ** з індексами 5, 6, ..., 20 наведено нижче:

а[1], а[2],..., а[10], с1ідП[5], сііДП[6], ..., сІідП[20].

Значення елементів масиву змінюються так само, як і значення інших змінних. Наприклад, для першого елемента масиву **а** це можна зробити операцією присвоєння **а[1]:=5** або операцією введення даних **геасЛп(а[1]).**

Застосовуючи операції вибору елемента та модифікації його значення, можна розв'язати досить широкий клас простих задач з обробки одновимірних масивів, які можуть вважатися базовими операціями в контексті складніших задач. Такими базовими операціями є:

- введення та виведення масиву;
- ініціалізація масиву;
- копіювання масиву;
- пошук максимального або мінімального елемента;
- обчислення узагальнювальних характеристик (сум елементів, їх добутків);
- пошук заданого елемента;
- перестановка елементів або обмін значеннями між елементами масиву;
- вставка та видалення елемента.

Базові операції обробки масивів зручно реалізовувати у вигляді процедур, що згодом можуть бути використані як «архітектурні блоки» при розв'язанні більш складних задач. Серед таких задач найважливішою є задача упорядкування масивів. Декілька алгоритмів розв'язання цієї задачі буде розглянуто в розділі 7.1.3, а решту розділу 7.1.2 присвятимо розгляду базових операцій обробки одновимірних масивів.

Введення та виведення масиву

Мова РазсаІ не має засобів введення та виведення масиву як цілісного об'єкта, ця операція виконується *поелементно* за допомогою оператора циклу. Під час введення елементів масиву необхідно врахувати те, що їх кількість, тип та тип індексів задаються в оголошенні масиву до початку виконання програми і не можуть бути змінені. Якщо межі індексів масиву точно не відомі, їх добирають так, щоб введена кількість елементів масиву під час виконання програми не перевищувала верхньої межі індексу. Наприклад, після оголошення масиву **а:** аггауСІ**. .100]** оТ **геаі** кількість елементів, що до нього вводяться, не повинна перевищувати 100.

Приклад 7.2_

Розглянемо реалізацію операцій введення та виведення одновимірного масиву. Для зберігання значень елементів масиву на етапі компіляції виділяється оперативна пам'ять, обсяг якої дорівнює означеній кількості елементів, помноженій на обсяг пам'яті, гцо її потребує збереження одного елемента. Під час виконання програми користувач вводить реальну кількість елементів, яка не повинна перевищувати оголошеної кількості. Наведений нижче код ілюструє принцип використання операції введення та виведення масиву. Зазначимо, що елементи масиву слід вводити через пробіл, оскільки при цьому застосовується оператор reaci.

```
ргодгаш ex7_2:
уаг
  та5:array[1..10] оТ іпіедег;
  п:inРедег: (кількість елементів масиву}
  і:іпі:едег; {поточний індекс}
Ьеділ
  мгіїе1п('Enier пиmber oT eietenis <=10');
  reac!(n);
  мгі1е1п('EnPer eietenis уаіиез ');
  Тог i: =1 Ро п сіо
    reaci (шasЦj ]);
  мгile1n('EnPereci array');
  Тог i :-1 іо п сіо
    мгіі:е(та5[і],'  '):
  мгіїеіп;
епсі.
```

Ініціалізація масиву

Введення значень елементів із клавіатури фактично є одним із способів ініціалізації масиву. Іншій спосіб ініціалізації масиву полягає у присвоєнні кожному його елементу деякого значення. Найбільш ефективно ця операція виконується за допомогою оператора Тог. Наприклад, у наведеному нижче коді десяти елементам масиву агг присвоюються квадрати цілих чисел від 1 до 10.

```
Тог i :=1 їо 10 сіо
   arr[i 3:=5чг(i);
```

Одновимірні *масиви-константи* записуються за наведеним нижче зразком як перелік значень їх елементів:

```
сопзї a:array[1..5] оТ іпіедег - (1,3,2,-5.6);
```

Така ініціалізація еквівалентна серії присвоювань

```
a[1]:=1;  a[2]:=3;  a[3]:=2;  a[4]:=-5:  a[5]:=6;
```

Для однотипних масивів **A** та **B** як для цілісних об'єктів визначена операція присвоєння **A : = B .** Однотипність масивів означає, що вони мають однакові типи індексів та однакові типи елементів. **У** результаті виконання операції присвоєння **A : = B** значення елементів масиву **B** присвоюються відповідним елементам масиву **A,** тобто здійснюється копіювання масиву **B** до масиву **A.**

Пошук максимального та мінімального значень

Розкриємо ідею найпростішого способу знаходження максимального значення у масиві (мінімальне значення можна знайти аналогічним способом). Деякій змінній, скажімо, змінній **шах,** присвоюється значення першого елемента масиву. Після цього виконується цикл, що послідовно порівнює значення кожного елемента, починаючи з другого, із поточним значенням змінної **шах.** Якщо значення елемента перевищує **шах,** воно присвоюється змінній **шах.** Отже, на кожній ітерації циклу у змінній **шах** міститиметься найбільше значення з пройденої частини масиву, а по завершенні циклу змінна **шах** зберігатиме максимальне значення в усьому масиві.

Приклад 7.3_

У деяких видах спортивних змагань виступ спортсмена оцінюється декількома суддями. Із сукупності виставлених ними балів вилучаються найвищий та найнижчий бали. На основі решти балів обчислюється середнє арифметичне, яке і зараховується спортсмену як оцінка його виступу. Необхідно за виставленими суддівськими оцінками визначити середню оцінку виступу спортсмена. Кількість суддів не перевищує десяти.

Оцінки суддів вважатимемо елементами масиву дійсних чисел. Кількість суддів, а значить, і кількість оцінок вводитимемо до змінної **п.** Якщо знайдено максимальний та мінімальний бал (змінні **шах і** ті **п)** та накопичено суму балів **у** змінній **зиш,** то за формулою **гез = (зиш - тіп - тах)/(п - 2)** можна обчислити шукане значення. Цю задачу розв'язує програма **ех7_3.** Результати її роботи зображено на рис. 7.1.

```
ргодгаш ех7_3;
изез сгі;
маг
   шагк:аггау[1..10] оТ геаі;{масив ОЦІНОК суддів
   і ,п:іпі;едег;              {індекс оцінок та їх кількість
   шіп.шах,                   {мінімальна та максимальна оцінки
   5иш                        {сума оцінок суддів
   гезиН:геа1;                {середнє арифметичне оцінок
Ьеділ
  сігзсг:
  мгіle1п('дгасіе сleTіпіпд');
  мгіїе1п('епїег пишьегз оТ агЬіїегз (<=10)');
  геасі(п);
  мгіі:е1п('епіег '.п,' дгасіез'):
  Тог і:=1 їо п сіо                {цикл уведення масиву оцінок}
    геасі(тагк[і ]);
  шіп:=шагк[1];                    {ініціалізація мінімальної.}
  шах:=шапк[1];                                {максимальної}
  зиш:=шапк[1];                          {та сумарної оцінки}
  Тог і:=2 їо п сіо {пошук мінімальної та максимальної оцінок}
  Ьеділ
    іТ шіп>шагк[і] ІЬеп шіп:=шагк[і];     {модифікація
                                           поточного мінімуму}
    іТ шах<шапк[і] ТНеп шах:=шагк[і];     {модифікація
                                           поточного максимуму}
```

```
    зит:=зит+тагк[i];              {підсумовування всіх оцінок}
  епсі;
  игііе1п('шагдіп дгасіез');
  и/rile1n( 'тах=' ,шах:3:2,¹ тіп=',тіп:3;2);
  гезии:=(зит-тіп-шах)/(п-2);
  мгіі:е1п('гегіїїі: дгасіе='.гезиі 1:3:2);
епсі.
```

```
                                        ШшШ        яшт
дгайе  iiei ininд                                      *i|
enier  питьегз      aiьilerї  (<=10)

enier i» дгайез
7 5 4 8
тагдіп  дгайеї
тах=8.00 тіп=4.00
гегиіі  дгайе=6.08
                                                       ДІ
```

РИС. 7.1. Результати роботи програми ex7_3. Обчислення середньої оцінки виступу спортсмена

Під час пошуку найбільшого або найменшого елемента масиву може виникнути потреба у визначенні його індексу. Значення індексу, як правило, використовується при подальшій перестановці елементів масиву, їх видаленні тощо. Для розв'язання цієї задачі застосований у прикладі 7.3 алгоритм потрібно модифікувати. А саме, окрім змінної тах (тіп) слід використати змінну, в якій зберігатимуться значення індексів. Припустимо, це буде змінна **лот.** Цій змінній спочатку присвоюється індекс першого елемента масиву, а в тілі циклу вона змінює значення тоді, коли і змінна тах (тіп). Отже, змінній **пот** присвоюється значення індексу того елемента, який виявився більшим за поточне значення тах (меншим за поточне значення тіп).

Приклад 7.4_

У магазині утворилася черга з декількох покупців. Відомий час обслуговування продавцем кожного покупця. Визначити час перебування кожного покупця у черзі, а також номер покупця, обслуговування якого потребує найменше часу.

Легко побачити, що час перебування кожного покупця у черзі дорівнює сумарному часу обслуговування його та всіх попередніх покупців. Якщо позначити час обслуговування і-го покупця змінною а час його перебування в черзі — *зещ,* то це значення визначається за формулою *зещ* = $+ i_2 + ... +$ Можна застосувати рекурентну формулу, згідно з якою час перебування покупця в черзі визначається як сума часу його обслуговування та часу перебування в черзі попереднього покупця: *вер)і - зеги^і +* Якщо час обслуговування *п* покупців подати у вигляді «-елементного масиву, то номер покупця з мінімальним часом обслуговування - це індекс мінімального елемента в масиві. Програмне розв'язання цієї задачі наведено нижче.

```
рrодгаш ex7_4:
Ш5€5 crP;
```

```
yar
    n,                          {КІЛЬКІСТЬ покупців                }
    i iinïедег:                 {ПОТОЧНИЙ індекс покупця           }
    i:array[1.,10]oT reai;      {масив часу обслуговування        }
    зеry:array[1.,10]oT reai    {масив часу перебування в черзі   }
    пот:inïедег;                {номер покупця з мінімальним
                                 часом обслуговування             }
    тin:rea1;                   {мінімальний час обслуговування   }
Ьеgin
  сіг5сг;
  гапсіоті ге:
  игile1n('cieTininд їИе питЬer оТ Ьиуег мііЬ а тінітит
           зеryice їіте');
  mrile1n('enier питЬer оТ їНе Ьиуегз (<=10)');
  reac)in(n);              {ввести кількість покупців
  Tor i :=1 їо n cio       {генерувати час обслуговування покупців
    ї[i] ;=гапсіот*10;
  мгіїе1n('зеryice їіте оТ їііе Ьиуегз');
  Tor i :=1 їо n cio
    мгіїе( Ї[I]:5:2.' ');
  мгіїein:
  зеry[1]:= ї[1]:    {час перебування у черзі першого покупця}
  Tor i :=2 їо n cio  {розрахувати час перебування у черзі     }
    5єгу[i]:=5єгу[i 1]+ї[i];
  мгіїе1n('ЗЄГУІСЄ їіте');
  Tor i :=1 їо n cio
    мгіїе(зеry[i]:5:2,' ');
  мгіїein;
.пот:=1;  {пошук покупця з мінімальним часом обслуговування}
  ті n:= £[1];
  Tor i :=2 їо n cio
    iT ї[i]<тіп Їlпеп
                Ьеgіп
                  тіп:=ї[i];
                  пот:=i;     {індекс мінімального елемента}
                enci;
  мгіїе('питЬer оТ Ьиуег Науіпд тіп ЗЄГУІСЄ їіте =');
  мгіїein(пот);
encl.
```

(IniacHne (ЮРАРАЗІЕХ?_4.III)

```
йеНпіпд іИе питЬer оГ Ьиуег мііЬ а ічіпіпиш зегиice їіте
enier питЬer о? Ibe Ьиуегї (<=10)
5
зегиice їіте оГ Ibe Ьиуегз
 0.89  1.31  0.92  8.91  1.83
зегиice їіте
 0.89  2.20  3.12 12.03 13.86
питЬer     Ьиуег Иаиіпд тіп зегиice їіте =1
```

РИС. 7.2. Результати роботи програми ex7_4. Визначення номера покупця з мінімальним часом обслуговування

Зауважимо, що кількість циклів і масивів, які входять до складу програми ex7_4, може бути зменшена. Спробуйте розв'язати задачу з прикладу 7.4, використовуючи якомога менше циклів та масивів.

Пошук у неупорядкованому та упорядкованому масивах

Пошук є однією з найбільш поширених задач програмування. Пошук в масиві за певним ключем полягає у визначенні номерів елементів масиву або **їх** значень, для котрих деяка умова, «ключ», виконується. Розрізняють задачі пошуку в упорядкованому та неупорядкованому масивах. В неупорядкованому масиві пошук можна здійснити лише за допомогою перегляду всього масиву. Такий пошук називається *лінійним.* Якщо значення елементів в масиві повторюються, то шляхом лінійного пошуку можна знайти лише перший з таких елементів, перервавши подальший пошук, або знайти усі потрібні значення, переглянувши весь масив. Приклади програм пошуку в неупорядкованому масиві першого з елементів і всіх елементів, які відповідають ключу пошуку, наведені нижче.

Приклад 7.5__

Створити масив за допомогою генератора псевдовипадкових чисел. Ввести з клавіатури деяке значення та визначити номер найпершого з елементів, що дорівнюють цьому значенню. Якщо таких елементів не існує, вивести відповідне повідомлення.

Лінійний пошук здійснюється шляхом перебирання всіх елементів масиву та порівняння кожного з них із заданим значенням. Якщо елемент знайдено, цикл пошуку переривається і виводиться знайдений індекс. А якщо цикл пошуку дійшов останнього елемента, і цей елемент не дорівнює заданому значенню, виводиться повідомлення про відсутність шуканого елемента. Описаний алгоритм пошуку елемента в неупорядкованому масиві реалізовано в програмі ex7_5, код якої наведено нижче. На рис. 7.3 зображено результати роботи програми ex7_5.

```
рrодгат ex7_5;        {пошук у неупорядкованому масиві        }
и$є5 сгї;
маг п.і:іпїедег:      {кількість елементів масиву та їх індекси}
    а:array[1..10] оТ іпїедег;          {вхідний масив}
    уа!ие:іпїедег;                      {шукане значення}
Ьеді п
  сігзсг;
  гапсіошіге;{ініціалізація генератора псевдовипадкових чисел}
  мгїїе1п('сіеТіпіпд їііе пишЬег оТ їііе діуеп сотропепї');
  мгїїе1п('епїег пипЬег оТ Ibe сошропепіз (<=10)');
  reacПn(n);
  Тог i:=1 їо n сіо                 {генерація масиву       }
    a[i] :=ranciот(10);
  «гїїе1п('депегаїесі array');
  Тог i ;=1 їо n сіо                {виведення масиву       }
    игїїе( a[i ],'  ');
  мгїїеіп;
  игїїе1п('епїег уаіие Тог зеагсіїї');
  reaciin(yaiue);                   {введення ключа пошуку  }
```

```
{                -. . . . . .-. . . . . .-. . . . . .                }
  Tor 1:=1 їо n cio                  (пошук першого елемента,    }
    IT a[i]=ya!ue їііеп              (що відповідає ключу пошуку}
                  Ьеділ
                    мгіїе1п('пот=',i);
                    Ьгеак;           (переривання циклу пошуку}
                  епсі;
  iT a[i]<>ya1ue їЬеп мгіїе1п('уаіие пої ТоипсГ ):
{--                                                              }
  reacПп;
епсі.
```

```
                                                              ш
йеГіпіпд Яіе питЬег     Иіе дімеп сопропепі
enier питЬer o? 1(ie сотропепіз (<=10)
7
депегаїей array
2 7 5 1 2 И 9
enïer иаіие ?or зearcИ
1
поп=4
                                                              А
```

РИС. 7.3. Результати роботи **програми ex7_5.** Пошук у неупорядкованому масиві **першого** з елементів, що відповідає **ключу пошуку**

Приклад 7.6_

Розглянемо задачу пошуку в неупорядкованому масиві всіх елементів, що їх значення відповідає заданому ключу. Алгоритм розв'язання цієї задачі відрізняється від алгоритму розв'язання попередньої задачі тим, що пошук не переривається, якщо знайдено перший з потрібних елементів. Цикл завершується, коли переглянуто всі елементи масиву. Використання змінної булевого типу, яка перед виконанням циклу пошуку має значення Таїзе, а коли елемент знайдено, набуває значення їгие, дає можливість визначити результат пошуку. Для розв'язання цієї задачі слід замінити у програмі ex7_5 оточений пунктирними лініями цикл пошуку елемента на нижченаведений код та додати оголошення змінної Лад. Модифікованій в такий спосіб програмі ex7_5 дамо назву ex7_5A.

```
  Лад:=Та1$е;                    (ознака успішності пошуку}
  Tor i:=1 їо n cio               (пошук значення у масиві}
    iT a[i]=ya!ue їЬеп
                  Ьеділ
                    мгіїе1п('потп=' .i);
                    Т1ад:=їгие;      (пошук вважати успішним}
                  епсі;
  iT пої Лад їЬеп мгіїе1п( 'Уаіие пої ТоипсГ);
{                                                  -. . . .}
```

```
йеЯпіпд ІГіе питЬег о(= Їііе діиеп сотропепї
іепїег питЬег о? іііе сотропепі^ (<=10)

ідепегаїей array
9 1 2 8 9 9 1
enier иаіие ґог 5earcй
9
пот=1
поп=5
пот=6
```

i .

РИС. 7.4. Результати роботи програми ex7_5A. Пошук у неупорядкованому масиві всіх елементів, що відповідають ключу пошуку

Найвідомішим методом прискорення пошуку заданого елемента в упорядкованому масиві є метод *бінарного пошуку*, що має також назву методу половинного ділення. Якщо масив упорядковано, то половину його елементів після порівняння шуканого значення із середнім елементом можна відкинути. Коли ці значення рівні, вважаємо, що шуканий елемент знайдено. Якщо еталонне значення менше, ніж значення середнього елемента масиву, то шуканий елемент може бути лише серед елементів лівої частини масиву, до котрої застосовується той самий метод, якщо — більше, то пошуковий метод застосовується до правої частини масиву. В результаті бінарного пошуку досліджується не більше ніж $[1o§_2\ n]$ елементів, де n — кількість елементів масиву, а квадратними дужками позначено цілу частину числа. Тому цей метод працює значно швидше за лінійний пошук.

Приклад 7.7_

Розглянемо задачу пошуку в упорядкованому масиві. Оголосимо змінні IeTI, гідHI та ті сісії е типу Ьуіе для позначення індексів першого (ліва межа), останнього (права межа) і середнього елементів ділянки масиву, що аналізується. Нехай масив елементів цілого типу позначено змінною таз. Кількість задіяних елементів масиву користувач вводить до змінної п. Використовуватимемо рекурсивну процедуру бінарного пошуку. Умовою припинення рекурсивного виклику процедури буде умова завершення пошуку (шукане значення збігається із значенням деякого елемента масиву) або умова відсутності шуканого значення в масиві (номер першого елемента ділянки масиву, що аналізується, перевищує номер останнього).

Алгоритм бінарного пошуку в упорядкованому масиві

1. Ввести розмір масиву.
2. Ввести елементи масиву та ключ пошуку.
3. Покласти ліву межу масиву рівною одиниці, праву межу - рівною введеній кількості елементів.
   3.1. Якщо ліва межа масиву більше правої, то завершити пошук, вважаючи його безрезультатним, інакше виконати дії, зазначені в пунктах 3.2-3.4.

3.2. Визначити індекс середнього елемента за наведеною нижче формулою

mШIe = (Ie/i + гіфі) **ciV 2.**

3.3. Порівняти ключ пошуку зі значенням середнього елемента. Якщо ці значення збігаються, то завершити пошук, вважаючи його успішним, інакше визначити підмасив, в якому треба продовжити пошук. Для цього перейти до пункту 3.4.

3.4. Якщо ключ пошуку менше значення середнього елемента, виконати процедуру бінарного пошуку для лівого підмасиву з межами *Ie/ї* та *micШe-1,* інакше виконати процедуру бінарного пошуку для правого підмасиву з межами *micIcIIe*+1 та *гіфі.*

4. Якщо пошук завершено успішно, вивести знайдений елемент та його індекс, інакше вивести повідомлення про безуспішність пошуку.

Цей алгоритм реалізовано програмою ex7_6. Результати роботи програми зображено на рис. 7.5.

```
Рrодгат ex7_6;          {бінарний пошук в упорядкованому масиві}
изез сгї;
уаг шаз:array[1..15] ої іпїедег:                {ВИХІДНИЙ масив}
    і.п:іпїедег;                  {індекс і кількість елементів}
   •Т1адЬоо1еап;                    {ознака успішності пошуку}
    х:inРедєг;                                        {ключ пошуку}
    тісІсПе:inРедег;              {індекс середнього елемента}

рrосесііиге Bin5earcIт(1eTi.гідlіі;:іпі;едег);
{рекурсивна процедура бінарного пошуку,
          параметрами є індекси лівого та правого елементів}
Ьеділ
  іТ 1eTI>гідHI                              {умова виходу з рекурсії}
     IЬеп Яад:=Та1зе                          {елемент не знайдено}
     еізе
       Ьеділ
         тісІсl1e:=(1eT1:+гідІі£) сііу 2;
         іТ х=таз[тісісПе]                   {умова виходу з рекурсії}
              іМеп Яад:=1;гие                {елемент знайдено}
              еізе іТ х<таз[тісісЛе]
                       їііеп Bin5earcH(1eіі;.тісІсі1e-1)
                       еізе BinЗearcИСтісісIIe+1.гідігР);
       епсі;
  епсі;

Ьеділ                                        {основна програма}
  сігзсг;
  мгііе1п('Ьinary searcH');
  мгіі:е1п('епїег питЬег оТ сотропепїз');
  reaciin(n);
  ^riPeїn('enier array');
```

```
  Tor i: =1 їо n сіо                    (введення елементів масиву}
     reacі(таз[i ]);
  мгїіе1п('епїег уаіие їо зеагсїї');
  reacПп(x);                              (введення ключа пошуку}
  Віп5еагсИ(1,п);
  iT Тіад їНеп мгїіе1п('тїет='.шісісїіе)
          еізе мгїіе1п( 'уаіие пої Тоипсl');
  reacї П;
епсі.
```

```
ЦС:\ВР\РА5\ЕХ7_6.EXE
Ьinar-y 5earcЬ
enier питЬer o? сотропепІБ
9
enier array
1 3 3 4 7 8 9 10 12
enier чаїие їо їearcK
9
l1en=7
```

шш

РИС. 7.5. Результати роботи програми ех7_6.
Бінарний **пошук в упорядкованому масиві**

Перестановка елементів масиву

Перестановка або обмін значеннями між двома елементами масиву здійснюється аналогічно до обміну значеннями між двома змінними (рис. 7.6).

1. Значення одного елемента масиву зберігається в допоміжній змінній.
2. Значення іншого елемента присвоюється першому елементу.
3. Значення допоміжної змінної присвоюється другому елементу.

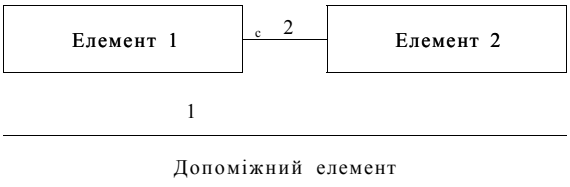


Рис. 7.6. Обмін значеннями між двома елементами масиву

Запишемо програму перестановки першого та другого елементів масиву.

```
уаг таз:array[1..10] оТ іпїедег;          (вхідний масив}
    їтр:іпїедег;                          (допоміжна змінна}
Ьеgіп
  шаз[1]:=5;                               {вхідні значення}
  таз[2]:=7;
  їпір:=таз[1]; (обмін значеннями через допоміжну змінну}
  таз[1]:=таз[2];
  шаз[2]:=їтр;
епсі.
```

Приклад 7.8_

Як приклад застосування методу перестановки елементів розглянемо задачу «перевертання», або інвертування, масиву: переставити значення першого і останнього елемента, другого і передостаннього тощо. Особливість задачі полягає в тому, що виконання операцій перестановки лише для лівої половини елементів здійснить інвертування всього масиву.

Алгоритм задачі інвертування масиву

1. Ввести *n* — розмір масиву та згенерувати значення його елементів.
2. Починаючи з першого елемента і до середини масиву виконувати обмін значеннями між *i*-м та (и-г+1)-м елементами масиву.
3. Вивести масив.

```
Prodgash ex7_7;                          {інвертувати масив}
Var ша$:array[1..10] oT iпїедег;         {вхідний масив}
    i.n:inPедег:    індекс поточного елемента та кількість
                                                 елементів}
    їтр:inїедег;                         {допоміжна змінна}
Ьеdіп
  мгїіе1п('Array inyerзion');
  мгїіе1п('Enїer питber oT eiешепіз');
  reaciin(n):
  мгїіе1п('бепегаїесі array:');
  гапсіошіге;        {ініціалізація генератора випадкових чисел}
  Tor i :=1 їо n cio          {генерація псевдовипадкових чисел}
    шаз[i] :=гапсіот(30);
  Tor i:=1 їо n cio             {виведення згенерованого масиву}
    иг1їе(таз[i],'  ');
  мгїїein;
  Tor i:=1 їо n ciiV 2 cio                  {інвертування масиву}
  Ьеdіп
    їтр:=ша$[i];
    шаз[i]:=шаз[n-i+1];
    шаз[n-i+1]:=їтр;
  епсі;
  мгїiein('ІПУЄГЇЄСІ array:');
  Tor i:=1 їо n cio             {виведення інвертованого масиву}
    мгїіе(гпаз[i],'  ');
  reaciin;
епсі.
```

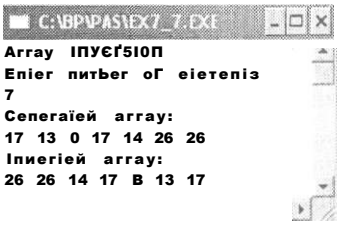


РИС. 7.7. Результати роботи програми ex7_7. Інвертування масиву

Вставка та видалення елемента, циклічний зсув елементів масиву

Під час розв'язання багатьох задач виникає необхідність вставити новий елемент на задану позицію в масиві (рис. 7.8, *а),* видалити заданий елемент масиву (рис. 7.8, 6) або циклічно зсунути елементи масиву на одну позицію (рис. 7.9). На рис. 7.8 та 7.9 елементи, що зсуваються, оточені жирною межею, а елементи, що видаляються або додаються, пофарбовані сірим.

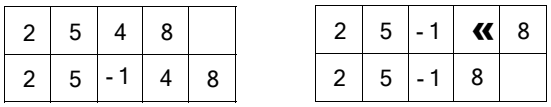


Рис. 7.8. **Вставка (а) та видалення (б) елементів масиву**

Перед тим як вставляти до масиву новий елемент, для нього слід звільнити місце. Це можна зробити шляхом переміщення на одну позицію вправо підмасиву, що починається з позиції, у яку буде вставлено новий елемент. Так, якщо необхідно вставити новий елемент Ітр на позицію 3 в (п- 1)-вимірний масив а, то підмасив а[3], ..., а[п-1] треба зсунути вправо на одну позицію. Переміщення підмасиву слід здійснювати поелементно, починаючи від його правого елемента і рухаючись до лівого (якщо зсув здійснювати зліва направо, то весь підмасив буде заповнений значенням елемента а[3]).

В результаті вставки кількість елементів масиву збільшиться. Згадаймо, що одна з властивостей масиву полягає у незмінності його розмірності. Тому кількість елементів у масиві можна збільшити лише в межах тієї розмірності, яка була вказана під час його оголошення. Наведемо фрагмент програми, який демонструє вставку до масиву нового елемента Ітр в задану позицію і.

```
mrileЩ'enier yaiue atci rozi 11 on');
reaciïn(Ртр.,]);
Tor i:=n ciомпїо з cio
  та$[i+1]:=та5[i]:                    {зсув підмасиву вправо}
шаз[Л:=1шр;
п:=п+1;                  {збільшення кількості елементів масиву}
Tor i:=1 їо п cio
  mrile(ша$[i].'  ');
```

Під час видалення певного елемента з масиву слід зсунути на одну позицію вліво частину масиву, що розташована праворуч від цього елемента. Так, якщо необхідно видалити третій елемент п-вимірного масиву, то підмасив а[4],..., а[п] треба зсунути вліво на одну позицію. Переміщення підмасиву слід здійснювати поелементно, починаючи від його лівого елемента і рухаючись до правого (якщо зсув здійснювати справа наліво, то весь підмасив буде заповнений значенням останнього елемента масиву). Після видалення кількість елементів масиву необхідно зменшити на одиницю. Наведемо фрагмент програми, який демонструє видалення ;-го елемента масиву.

```
мгіїе!п('enier розіїіоп Tor cieieїe');
reacЛп(л);
Tor i :*\i їо п-1 сіо
  таз[i]:=шаз[i+1];                    {зсув підмасиву вліво}
п:=п-1;                  {зменшення кількості елементів масиву}
Tor i :=1 їо п сіо
  мгіїе(шаз[i],'  ');
```

На відміну від операцій видалення або вставки елементів, під час циклічного зсуву масиву кількість його елементів не змінюється. Для циклічного зсуву вправо (рис 7.9, *а)* слід зберегти в резервній змінній останній елемент та на одну позицію вправо перемістити підмасив, що починається з першого елемента і завершується передостаннім. Після цього необхідно записати на місце першого елемента той, який раніше був останнім. Коли виконують циклічний зсув вліво (рис. 7.9, *б),* у резервній змінній зберігають перший елемент, зсувають на одну позицію вліво підмасив, що починається з другого елемента і завершується останнім, та записують на місце останнього елемента той, який раніше був першим.

5	-1	4	8	2
2	5	-1	4	8

а

5	-1	4	8	2
-1	4	8	2	5

б

Рис. 7.9. Циклічний зсув елементів масиву вправо (а) та вліво (б)

Наведені нижче фрагменти програми демонструють циклічний зсув елементів масиву вправо та вліво на одну позицію.

```
«гіїе1п('гідіїї зНіТї');    {циклічний зсув елементів вправо}
їшр:=шаз[п];                  {останній елемент стає першим}
Tor i:=п-1 сіомпїо 1 сіо
  шаз[i+1]:=шаз[i];
таз[1]:=їшр;
мгіїе1п(' IeTї зНіТї');     {циклічний зсув елементів вліво}
їтр:=таз[1];                  {перший елемент стає останнім}
Tor i :=1 їо п-1 сіо
  таз[i]:=таз[i+1];
таз[п]:=їтр;
```

7.1.3. Сортування масиву

Однією З найбільш поширених операцій обробки масивів є їх упорядкування, або сортування. Упорядкування масиву — це зміна порядку розташування його елементів за певним критерієм. Наприклад, числовий масив можна упорядкувати за зростанням значень його елементів або за їх спаданням, а масив рядків можна відсортувати в алфавітному порядку. Найчастіше сортування масиву здійснюється з метою полегшення подальшого пошуку.

Відомо багато методів сортування масиву, що відрізняються швидкодією й обсягом оперативної пам'яті, яка при цьому використовується. Серед цих методів можна вирізнити методи внутрішнього та зовнішнього сортування. Методи внутрішнього сортування не передбачають використання допоміжних масивів. Ці методи застосовують до масивів, що повністю розташовані в оперативній пам'яті. Методи зовнішнього сортування застосовують до великих масивів даних, які зберігаються на зовнішніх носіях. У цьому розділі розглянемо методи внутрішнього сортування масиву, обмежуючись задачею сортування за зростанням. Методи внутрішнього сортування прийнято поділяти на дві групи: елементарні (прямі) та удосконалені методи.

Найбільш відомими елементарними методами сортування масиву є:

- сортування вставкою (включенням);
- сортування вибором;
- сортування обміном (бульбашкове сортування).

З удосконалених методів сортування найчастіше використовуються такі:

- швидке сортування, або метод Хоара;
- сортування включенням зі спадним приростом, або метод Шелла;
- сортування за допомогою дерева, або пірамідальне сортування;
- сортування методом злиття.

Нижче буде розглянуто всі три вищезгаданих елементарних методи сортування та два удосконалених — метод Хоара й метод злиття.

Сортування методом вставки

На кожному кроці цього методу масив розділений на дві частини: ліву, вже відсортовану, та праву, ще не відсортовану. Перший елемент правої частини вставляється до лівої частини так, щоб ліва частина залишалася відсортованою. У результаті відсортована частина збільшується на один елемент, а невідсортована — на один елемент зменшується. Отже, на кожному кроці алгоритму сортування методом вставки слід виконати дві операції: пошук позиції для вставки елемента та власне його вставку із подальшим зсувом на одну позицію вправо від елементів відсортованої частини. Цей зсув «затре» перший елемент невідсортованого підмасиву останнім елементом відсортованого. Спочатку відсортованим підмасивом вважаємо перший елемент, а решту елементів масиву відносимо до невідсортованої частини.

Приклад 7.9

Формалізуємо алгоритм сортування методом вставки, а також наведемо його програмну реалізацію. Підмасив, що містить елементи з другого до останнього, вважати невідсортованою частиною. Поки ця частина не стане порожньою, виконувати такі дії.

1. Зберегти перший елемент невідсортованого підмасиву в допоміжній змінній.
2. Визначити позицію вставки збереженого елемента у масив. Для цього дотримуватися перелічених далі вказівок.

2.1. Вважати перший елемент масиву поточним.

2.2. Доки елемент для вставки більше за поточний, збільшувати індекс поточного елемента.

3. Вставити збережений на кроці 1 елемент на знайдену позицію вставки, зсунувши на одну позицію вправо решту відсортованої частини.

4. Пересунути початок невідсортованої частини на одну позицію вправо.

```
рrодгаш ex7_8;                                  {сортування вставкою}
изез crї;
уаг n.1.з,к:ілредег:                {кількість та індекси елементів}
    а:array[1..10] оТ іпїедег;
    їтр;іпїедег;    {елемент, ДО вставляється у відсортовану
                                                частину масиву}
Ьеді п
  сігзсг;
  гапсіоті 2е:      {і ні ці алі зу вати генератор випадкових чисел}
  мгііе1п('іпзегїіоп зогї');
  мгїїе!п('епїег питЬег оТ їііе сотропепіз (<=10)');
  reaciin(n);                  {ввести кількість елементів масиву}
  Тог i:=1 їо n сіо                                {генерувати масив}
    a[i] :=гапсіот(30);
  мгїіе1п('депегаїесі array');
  Тог i:=1 їо n сіо                  {вивести згенерований масив}
    мгііе(а[i].'  ');
  мгїіеіп;
  мгїіе1п('зогї processз');
  Тог i :=2 їо n сіо                   {сортувати методом вставки}
  Ьеді п                {і - початок невідсортованого підмасиву}
    їшр:=а[i];                        {вибрати елемент для вставки}
    3;=1;                             {цикл пошуку позиції вставки}
    кііііе Рщр>а[л] сіо             {якщо елемент, що вставляється,}
      3:=3+1:                          {менший або рівний поточному,
                                         то з фіксує позицію вставки}
    Тог к:=i-1 сіомпїо з сіо             {зсунути вправо елементи}
      а[к+1]:=а[к]:                      {невідсортованої частини}
    а[3]:=їшр;           {вставка вибраного елемента у позицію з}
    Тог к:—1 їо n сіо              {виведення промінних результатів}
      мгііе(а[к].'  ');
    игіїеіп;
  епсі;
  мгїіе1п('зогїесі array');
  Тог i; =1 їо n сіо                {виведення відсортованого масиву}
    мгїіе(а[i].'  ');
  геасПп;
епсі.
```

```
IП5ЄГПЮП 50ГІ
enier пишЬer оГ lИe сотропепїз (<=10)

депегаїей array
13 29 8 2В 17
50ґІ рГ0С€55
13 29 0 20 17
0 13 29 20 17
0 13 20 29 17
0 13 17 20 29
•.(«1-І є;) array
0 13 17 20 29

«І І .... .
```

РИС. 7.10. Результати роботи програми ex7_8. Сортування масиву методом вставки

Сортування методом вибору

Цей метод, як і метод сортування вставкою, розділяє масив на дві частини: ліву, вже впорядковану, та праву, ще не впорядковану. Вибирають найменший елемент невідсортованої частини. Цей елемент міняють місцями з її першим елементом, збільшуючи на одиницю довжину відсортованої частини масиву. Отже, на першому кроці алгоритму невпорядкованою частиною є весь масив, з котрого вибирають мінімальний елемент. Цей елемент міняють місцями з першим елементом масиву. На другому кроці невпорядковану частину масиву складають елементи від другого до останнього. Серед цих елементів вибирають найменший, котрий міняють місцями з другим. Процес триває доти, доки у невідсортованій частині не залишиться один елемент.

Приклад 7.10

Формалізуємо алгоритм сортування методом вибору і реалізуємо його мовою РазсаІ. Весь масив вважати невідсортованого частиною. Поки ця частина містить більше одного елемента, виконувати такі дії:

1. Вибрати перший елемент невідсортованої частини масиву і вважати його мінімальним; запам'ятати індекс цього елемента.
2. Для елементів від наступного після вибраного і до останнього повторювати такі дії.
   2.1. Порівняти вибраний елемент і поточний.
   2.2. Якщо вибраний елемент більший за поточний, запам'ятати поточний елемент як мінімальний, а його індекс — як індекс мінімального елемента.
3. Поміняти місцями мінімальний і вибраний на кроці 1 елементи.
4. Пересунути початок невідсортованої частини на одну позицію вправо.

```
ргодгат ex7_9;                    {сортування методом вибору}
изез сгї;
уаг п,і,,]:іпіедег;          {кількість та індекси елементів}
    а:array[1..10] оТ іпїедег;
    тіп.ішіп:іпїедег;    {мінімальний елемент 1 його індекс}
```

```
Ьеділ
  сіг5сг;
  гапсіошіге:      {ініціалізувати генератор випадкових чисел}
  игіїєіп('5єієсїі0п  зогї');
  мгіlеin('enier питber оТ liie сотропеніз (<=10)'):
  reaciin(n);                {ввести кількість елементів масиву}
  Тог i:=1 їо n сіо                              {генерувати масив}
    a[i]:=гапсіотС 30);
  мгііе1п('депегаїеci array'):
  Тог i:=1 їо n сіо                {вивести згенерований масив}
    и/гіle(a[i].'  ');
  игіїеіп;
  мгіle1п('зегіез оТ зеїесіїоп'):
  Тог 1 :=1 їо n-1 сіо
  Ьеділ
    шіп:=a[i]; {пошук мінімального елемента в діапазоні від
                                  і-го до останнього елемента}
    ішіп:-і:                    {індекс мінімального елемента]
    Тог 3:=i+1 їо n сіо          {пошук мінімального елемента}
      іТ шіп>a[Л Ібеп
                 Ьеділ
                   шіп:=a[Л:
                   і ті п:  :
                 епсі;
    a[ішіп]:=a[i]: {обмін місцями мінімального та поточного
                                                  елементів}
    a[i]:=шіп;
    Тог ,]:=1 їо n сіо        {виведення промінних результатів}
      м~іїе(аШ.'  '):
    игіїеіп;
  епсі;
  игіlеіп('зогїеci array');
  Тог i:=1 їо n сіо          {виведення відсортованого масиву}
    мгіїе(a[i],'  ');
  reaciin;
епсі.
```

```
5€І€СІІ0П 50Ґ1                                    ^
enier питber оГ IПe сопponeniз (<=10)
5
депегаїей array
16 12 11 28 6
5€Гі€5 ОГ 5€І€СІІ0П
6 12 11 28 16
6 11 12 28 16
6 11 12 28 16
6 11 12 16 28
ьогїeei array
6 11 12 16 28
    a                                              >г...
```

Рис. 7.11. Результати роботи програми ex7_9. Сортування масиву методом вибору

Сортування методом обміну

Базовою операцією в цьому методі є порівняння двох сусідніх елементів масиву. Якщо їх розташування суперечить умові впорядкування, вони міняються місцями. Послідовне застосування такої операції до всіх пар елементів масиву, від останньої пари до першої, дозволить виявити найменший елемент в першій позиції. Друга назва цього метода - бульбашкове сортування - пояснюється схожістю процесу обміну місцями сусідніх елементів зі спливанням більшої бульбашки. Під час сортування методом обміну впорядкованою буде ліва частина масиву, а щойно описаний процес повторюється для правої частини, котра на кожній ітерації методу зменшуватиметься на один елемент.

Приклад 7.11___

Розглянемо алгоритм сортування методом обміну та його реалізацію мовою РазсаІ.

1. Установити лічильник ітерацій рівним одиниці.
2. Для елементів масиву, від останнього до елемента з індексом, що дорівнює поточному значенню лічильника ітерацій, повторювати такі дії.
   2.1. Якщо поточний елемент більший за попередній, поміняти ці елементи місцями.
   2.2. Перейти до попереднього елемента.
3. Збільшити лічильник ітерацій. Якщо значення лічильника дорівнює кількості елементів масиву, завершити сортування.

```
ргодгаш ex7_10;                                  {сортування обміном}
изез сгї;
уаг п. ,л.к:InРедег;              {КІЛЬКІСТЬ та індекси елементів}
    a:array[1..10] оТ іпїедег;
    Ртр:іпїедег;                  {допомівкний елемент для обміну}
Ьеділ
  сігзсг;
  ranciotire;
  мгїіе!п('excїїanде зогї');
  мгїіе1п('enier питЬег оТ iiie сотропепіз (<=10)');
  reacПп(n);
  Тог i:=2 їо п сіо                              {генерувати масив}
    a[i] :=ranc1от(30);
  мгiie!п('деnегaieci array');
  Тог i:=1 їо п сіо                  {вивести згенерований масив}
    мгїіе( a[i].' ');
  мгїіеіп;
  мгїіе1п('зогї processз');
  Тог i:=2 їо п сіо                    {сортувати методом обміну}
    Тог з;=п сіоипїіо i сіо
      IT a Ш о и - I ] їііеп
                    Ьеділ          {поміняти елементи місцями}
                      Imp :=a[Л;
                      a Ш : - a [ M ];
                      a[i-1]:=iшp;
```

```
                {виведення проміжних результатів}
              Tor к:=1 їо п cio
                мгііе(а[к],' '):
              мгіїеіп;
            ЄПСі;
  мгііеіп(¹ зогіесі array'):
  Tor i:=1 їо п cio              {вивести відсортований масив}
    ирile(a[i],' ');
  геасПп;
епсі.
```

```
ехсііапде БОГІ
епкег питЬег    Ібе сотропеніз (<=10)
5
депеї'аіей array
В 26 14 9 17
50ГЇ рГ0СЄ55
0 26 9 14 17
0 9 26 14 17
0 9 14 26 17
0 9 14 17 26
50Пе<] array
0 9 14 17 26 _
```

РИС. 7.12. Результати роботи програми ех7 10. Сортування масиву методом обміну

Алгоритми сортування масивів методами вибору, вставки та обміну не потребують додаткової оперативної пам'яті. Час виконання розглянутих алгоритмів сортування пропорційний кількості операцій порівняння або перестановок елементів. Сортування «-елементного масиву методом вибору потребує $n^2/2$ операцій порівняння та n операцій обміну елементів. Метод сортування вставкою потребує $n^2/4$ операцій порівняння та стільки ж операцій обміну, а метод сортування обміном — $n^2/2$ операцій порівняння та $п^2/2$ операцій обміну. Отже, методи вставки та вибору за характеристиками приблизно еквівалентні, проте метод обміну по • ступається перед ними швидкодією.

Удосконалені методи сортування масиву використовують значно меншу кількість операцій порівняння та перестановок елементів. Детальний аналіз алгоритмів сортування зроблено у [5, 12, 13]. Розглянемо один з удосконалених методів сортування масиву.

Швидке сортування

Автор цього метода Ч. Хоар назвав його швидким сортуванням, оскільки для більшості масивів цей метод потребує приблизно $(n/6)$ 1о§ n обмінів елементів і n 1о§ n порівнянь, тобто набагато менше, ніж будь-який з елементарних методів. Метод функціонує за принципом «розділяй та пануй»: елементи масиву діляться на дві частини, і кожна з них потім сортується окремо. Для цього обирають деякий елемент x, назвемо його розділовим. Мета полягає у розташуванні всіх менших за x

елементів зліва від *x,* а всіх більших за *x* елементів - справа від *x.* Поділивши масив, слід повторити процедуру сортування для кожної частини, потім - для частин **цих** частин і т. д„ доки кожна з частин масиву не міститиме лише один елемент. Зауважимо, що у деяких модифікаціях методу Хоара розташування та значення розділового елемента можуть змінюватися під час розподілу елементів. Розглянемо одну з таких модифікацій [5].

Отже, алгоритм методу Хоара є рекурсивним і для його реалізації було б зручно застосувати рекурсивну процедуру. Параметрами цієї процедури будуть змінні IeTї та гідНІ, що позначатимуть ліву та праву межу розглядуваної частини масиву. Розділовим елементом вважатимемо середній за номером елемент частини масиву і присвоїмо значення цього елемента змінній x. Рекурсивний виклик процедури для поділу лівого підмасиву на дві частини здійснюється в тому разі, якщо ліва межа підмасиву менша за індекс його середнього елемента, а для поділу правого підмасиву - якщо права межа більша за індекс середнього елемента. Для поділу елементів на дві частини застосовуємо два цикли иігїїе у середині циклу гереаї-ипїіі. На кожній ітерації зовнішнього циклу здійснюватиметься один обмін елементами між лівою та правою частинами. Внутрішні цикли (здійснюють перегляд масиву зліва та справа) призначені для знаходження чергової пари елементів, що мають бути переставлені. Зазначимо, що переставленим може бути і сам розділовий елемент, при цьому він може втратити властивість розділовості. Цей факт не впливає на коректність роботи розглянутої далі програми.

Нижче наведено код програми ex7_11, а на рис. 7.13 зображено результати її виконання. Цикл, що здійснює виведення проміжних результатів, у код програми ex7_11 не включено.

Приклад 7.12

```
ргодгаш ex7_1i;                    {швидке сортування - метод Хоара}
изез сгї;
уаг і,п:іпїедег;     {кількість елементів масиву та їх індекси}
    а;аггау[1..10] оТ іпїедег;                  {вихідний масив}

ргосесііиге диіскзогї(1 еТї, гі дііїї: іпїедег);
 уаг і,і,к,х,їшр;іпїедег;     {розділовий елемент та допоміжні
                                змінні}
 Ьеді п
    =1еГї;                          {ліва границя підмасиву      }
    =гід!тЬ;                        {права границя підмасиву     }
    =а[(1еТт+гїдпї) сшу 2];         {значення бар'єрного елемента}
  мгіїе!п( 'leTї=" ,1eTї.' гідПї='.гідііїї.' х-'.х):
  repeaï                            {розділити масив на підмасиви}
     міліє а[ї]<х do і :=і+1;       {визначити індекси елементів,}
     мИПе а[Л>х do і;=1-1;          {цр обмінюються              }
     iT i<=i їйеп
            Ьеді п                  {обміняти елементи місцями   }
              їшр:=а[і];
              а[і];=а[1];
              а[і];=їшр;
              І:=І+1;
```

```
                        {виведення проміжних результатів}
                  Tor к:=1 їо п сіо
                    \л/rile(a[к].¹   '):
                  мгіїеin;
                епсі;
  ипїіі і>з;
  іТ 1eTI<] IНеп
    риискзогPCIeTPЛ);          {впорядкувати лівий підмасив}
  iT i<гідііl IЬеп
    диискзогКі.гідні);         {впорядкувати правий підмасив}
 епсі;

Ьеділ
  сігзсг;
  гапсіоті re; {ініціалізація генератора псевдовипадкових чисел}
  игileInC 'яиіск зогі');
  мrile1п('enier питЬer oT Ilie сотропепіз (<=10)');
  reaciin(n);
  Tor i: =1 їо п сіо              {згенерувати масив          }
    a[i] :=ranciот(30);
  мrile!n('депегаїеci array');
  Tor i:=1 їо п сіо               {вивести згенерований масив }
    мrile(a[i],'  '):
  йгшєіп;
  игile!n('зогі процезз');
  циискзогКІ.п);
  мrile!n( 'зогіеci array');
  Tor i:=1 їо п сіо               {вивести відсортований масив}
    мrile(a[i],'  ');
  мгіїеin;
  reaciin;
епсі.
```

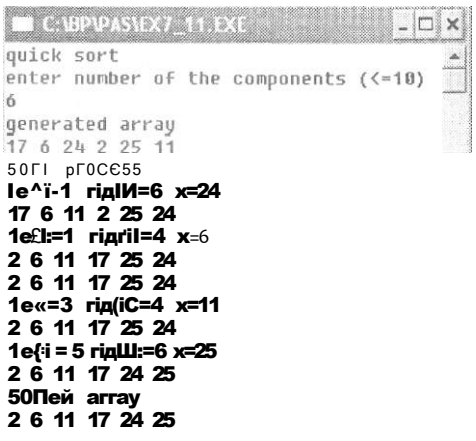


Рис. 7.13. **Результати роботи програми ex7_11. Швидке сортування масиву**

Сортування методом злиття

У запропонованому Дж. фон Нейманом методі сортування злиттям, як і в методі Хоара, реалізовано принцип «розділяй та пануй». Масив ділиться навпіл, до кожної половини застосовується рекурсивно та сама процедура сортування злиттям, а відсортовані частини з'єднуються в один впорядкований масив (рис. 7.13).

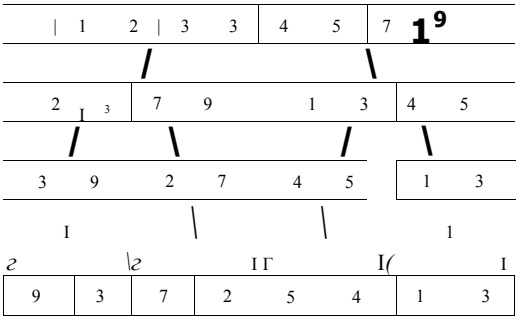


Рис. 7.14. Схема сортування методом злиття

Отже, базовою операцією методу є злиття двох упорядкованих масивів в один. Ефективний спосіб виконання цієї операції полягає в тому, хцо елементи масивів порівнюються і за результатами порівняння в новий масив записується елемент або з першого, або з другого масиву. Один з масивів під час злиття може закінчитися раніше. В такому випадку елементи іншого масиву, які ще не були опрацьовані, слід додати до нового масиву.

Час злиття упорядкованих масивів лінійно залежить від їх сумарної довжини. Враховуючи цей факт, неважко буде довести, що повне сортування методом злиття, як і сортування методом Хоара, потребує виконання 0(п lo§ п) базових операцій над елементами масиву розмірності *п.*

Програму сортування методом злиття наведемо у прикладі 7.13 з наступного розділу. Задачу злиття двох масивів реалізуємо за допомогою процедури. Оскільки процедура злиття має бути застосована до різних підмасивів, то зручно було б передавати масив до процедури як параметр. Особливостям використання масивів як параметрів підпрограм і присвячено наступний розділ.

7.1.4. Масиви як параметри

Масиви можна використовувати як параметри процедур і функцій. Загальні правила використання формальних параметрів і заміни їх аргументами (фактичними параметрами) застосовуються і до масивів. Масиви можуть бути передані у процедури як параметри-зиачення або як параметри-змінні. Коли масив передають як параметр-значення, у стековій пам'яті створюється його копія, і під час виконання підпрограми він не змінюється. Якщо масив передають як параметр-змінну, то у стекову пам'ять пересилається покажчик, тобто адреса першого елемента масиву. Це приводить до того, що під час виконання процедури сам масив змінюється, і його можна повернути з процедури для подальшого використання.

Можна рекомендувати передавати масиви як покажчики (параметри-змінні) за умови, що вони містять велику кількість елементів і потребують великого обсягу оперативної пам'яті для свого зберігання. У цьому разі стекової пам'яті, розмір якої становить 16 Кбайт, може не вистачити для збереження копії масиву.

Приклад 7.13_

Наведемо програмну реалізацію методу сортування злиттям. Саме сортування злиттям реалізуємо рекурсивною процедурою **$orі,** а злиття упорядкованих масивів - процедурою Мегде. Параметрами-значеннями процедури **Мегде** будуть масиви, які необхідно злити, та їх довжини. Параметром-змінною буде масив, до якого треба записати результат злиття. Крім того, знадобиться допоміжна процедура

CopyArr(a:Arr;x.y:іпїедег;уаг Ь:Arr;r:іпїедег)

що копіює частину масиву а, від елемента з номером **х** до елемента з номером **у,** до масиву **Ь,** а також процедура **Оиірії,** яка виводитимє до вікна програми вміст масиву, що передається їй як аргумент.

```
ргодгаш ех7_12;                    {сортування методрм злиття   }
изез сгі;
Iуре
  Arr=array[1..10] ої іпїедег; {ТИП масиву з 10 цілих чисел }

уаг i,n;іпїедег;            {індекс і кількість елементів масиву}
    а,д,е.Т:Arr;                    {а - масив, що сортується,
                                     д,е,Т - ДОПОМІЖНІ масиви     }

{злиття упорядкованих масивів а та Ь}
ргосесііиге Мегде(а,Ь;Arr;п,т:іпїедег;уаг с:Arr);
уаг
  i.a;іпїедег;
Ьеділ
  i:-1:  i:-1:
  мііііе (i<=n) апсі (з<=ш) сіо   {формувати вихідний масив    }
    і? a[i]<Ь[Я їЬеп              {якщо елемент масиву а менший}
    Ьеділ
      c[1+M3:-a[1];
      і:=і+1;          {перейти до наступного елемента масиву а}
    епсі
    еізе                          {якщо елемент масиву Ь менший}
    Ьеділ
      c[I+M]:=ЬШ;
      1 =,I+1;         {перейти до наступного елемента масиву Ь}
    епсі
  Тог i -i їо п сіо                  {якщо масив а не вичерпано}
    ci>i-1]:=a[i];
  їог ;i:=д їо ш сіо                 {якщо масив Ь не вичерпано}
    с[1+М]:-Ь[;)]:
епсі;

{копіювання частини масиву а до масиву Ь}
ргосесііиге CopyArr(a;Arr;x,y;іпїедег;уаг Ь;Arr;r:іпїедег);
```

```
уаг
  i:іпїедег;
Ьедіп
  Тог i: =х їо у сіо
    Ь[і-Х+2]:=а[і];
епсі;

{виведення масиву а, що мас довжину п}
ргосесіиге Оиїриї(а:Агг;п:іпїедег):
уаг
  і;іпїедег;
Ьедіп
  Тог і: –1 їо п сіо
    мгіїе(а[і].' ');
  мгіїеіп;
епсі;

{сортування частини масиву від і-го елемента до і-го}
ргосесіиге 5огї(і ,1;іпїедег);
уаг
  ї: іпіедег;
Ьедіп
  іТ і<з їНеп          {якщо сортується більше ніж один елемент}
  Ьедіп
    ї;=і + (3-і) СІІУ 2;     {середина частини, що сортується   }
    ІТ і-д-1 їНеп   {якщо сортується більше ніж два елементи}
    Ьедіп
      5огї(1,ї);             {сортування лівої ПОЛОВИНИ          }
      5огї(ї+1,з);           {сортування правої ПОЛОВИНИ         }
    епсі;
    СоруАгг(а,і,ї,д,1);        {копіювання відсортованих частин}
    СоруАгг(а,ї+1,з,е. 1);     {до резервних масивів д та е    }
    Мегде(д.е,ї-і+1.з-Ї.Т);        {злиття відсортованих частин}
    СоруАгг(Т,1.,і-і+1,а,і);
    игііе( 'Тhe зогїесі [' ,і, .".Д.'] рагї ');
    0иїриї(Т,;і-і+1);          {виведення проміжних результатів}
  епсі;
епсі;

Ьедіп
  сі г5сг;
  гапсіотіге;
  мгііе!п('шегде зогї');
  мгііе!п('епїег питЬег оТ їНе сотропепїз (<=10)');
  геасПп(п);
  Тог і :=1 їо п сіо                        {генерувати масив}
    а[і]:=гапсІот(30):
  мгііе!п('депегаїесі array')
  Оиїриї(а.п);                    {вивести згенерований масив}
  Богї(І.п);                                {сортувати масив}
  мгііеІпС' зогїесі array');
  Оиїриї(а.п);                   [вивести відсортований масив}
  геасііп;
епсі.
```

```
C:\BP\PAШX7_
тегде 50ГІ
епіег питЬеї' оР Ыіе сотропепїз (<=1В)
7
депегаїей аггау
22 28 25 8 6 28 27
Тґіе зогїей [1,2] раґї 22 28
Тґіе зогїей [3,4] раґї 8 25
ТПе 50ҐІЄЙ [1,4] раві 8 22 25 28
ТПе ЗОҐІЄЙ [5,6] раві 6 28
Тііе БОҐІЄЙ [5,7] раґЬ 6 27 28
Тііе зогіей [1,7] раґї 6 8 22 25 27 28 28
БОГЬЄСІ аггау
6 8 22 25 27 28 28
```

РИС. 7.15. Результати роботи програми ex7_12. Сортування злиттям

7.2. Багатовимірні масиви

Як було зазначено вище, одновимірні масиви застосовуються для зберігання послідовностей. Проте для багатьох структур даних зображення у вигляді послідовності є неприйнятним. Наприклад, результати матчів футбольного чемпіонату найзручніше подавати у вигляді квадратної таблиці. Для зберігання таких структур даних застосовують *багатовимірні масиви,* серед яких найбільш широко використовуються двовимірні масиви (матриці). У даному розділі розглядаються базові операції над елементами двовимірних масивів, а також техніка програмного розв'язування деяких матричних задач лінійної алгебри.

7.2.1. Оголошення багатовимірних масивів. Доступ до елементів

Розглянемо прямокутну таблицю з *mxn* однотипних елементів, що містить *m* рядків та *n* стовпців. У математиці таку таблицю називають *матрицею,* а дані, що їх містить матриця, — елементами. У програмуванні матричні структури називаються *двовимірними масивами.*

Наведемо синтаксис оголошення змінної матричного типу в мові РазсаІ:

<ім'я матриці>:аггау**[«нижній індекс рядка>..«верхній індекс рядка>,<нижній індекс стовпця>.,<верхній індекс стовпця>]** оТ **«тип елементів>:**

У цьому оголошенні **аггау, оТ** - зарезервовані слова, **<тип елементі в>** - будь-який тип, крім файлового, **<нижній індекс рядка>** і **«верхній індекс рядка>** — константи, що визначають межі діапазону допустимих значень індексу рядка, а **«нижній індекс стовпця>** та **«верхній індекс стовпця>** — константи, що визначають межі діапазону допустимих значень індексу стовпця. Масиви, що мають більш ніж два виміри, оголошуються в аналогічний спосіб.

Зазначимо, що, як і будь-який тип даних користувача, тип двовимірного масиву можна оголосити також в розділі їуре окремо від оголошення змінної матричного типу.

Наведемо приклади оголошень багатовимірних масивів. Нагадаємо, що обсяг сегмента пам'яті, де будуть зберігатися дані Разсаі-програми, становить 64 Кбайт, і тому під час визначення розміру масиву необхідно враховувати можливість переповнення пам'яті.

```
сопзї к=10; ш=20; п=5;
їуре  TMaPrix=array[1.,к.1..т] оТ inPедег:    {200 елементів}
маг   a:TMaïrix;
      Ь:array[1..п.1..п] оТ reai;             {25 елементів}
      с:array[1..4,1..3] оТ сИаг;             {12 елементів}
      c):aггayCЬyPє. 1. .2] оТ ініедег;       {512 елементів}
      таї:array[1..3,5..10,2..4] оґ ініедег;  {54 елементи, тривимірний масив}
```

Доступ до елементів матриці здійснюється операцією індексування [] за двома індексами, які визначають номер рядка та номер стовпця елемента. Синтаксис операції індексування є таким:

<ім'я масиву>[<номер рядка>,<номер стовпця>]

Зазначимо, що доступ до елементів масивів, які мають більш ніж два виміри, здійснюється також із застосуванням операції індексування. Але її операндами є більш ніж два індекси. Тепер дамо приклади індексування елементів оголошених вище масивів.

```
c[4.3]:='C';
шаl[2,6,3] :=a[к,ш]+cl[0,2];
```

Перейдемо до розгляду форм зображення багатовимірних масивів. Як вже зазначалось, природною формою зображення двовимірного масиву вважається таблиця. Для прикладу на рис. 7.16 зображено масив, що може бути оголошений як **a:array[1..4,1..4] оТ ініедег.**

Напрям зміни другого індексу

1

Напрям зміни першого індексу	1	[ІД]	[1,2]	[1,3]	[1,4]
	2	[2Д]	[2,2]	[2,3]	[2,4]
	3	[3Д]	[3,2]	[3,3]	[3,4]
	4	[4Д]	[4,2]	[4,3]	[4,4]

Рис. 7.16. Двовимірний масив розмірністю 4х4

Приклад зображення тривимірного масиву наведено на рис. 7.17.

Природне зображення багатовимірного масиву має суттєво відрізнятися від його зображення в оперативній пам'яті, адже оперативна пам'ять має лінійну структуру. За яким же принципом елементи матриці розташовуються в пам'яті

комп'ютера? Спочатку у послідовних комірках записуються всі елементи першого рядка, потім у подальших послідовних комірках записуються елементи другого рядка і т. д., поки не буде вичерпано всі елементи. Розмір оперативної пам'яті визначається при оголошенні багатовимірного масиву і не змінюється під час роботи з ним.

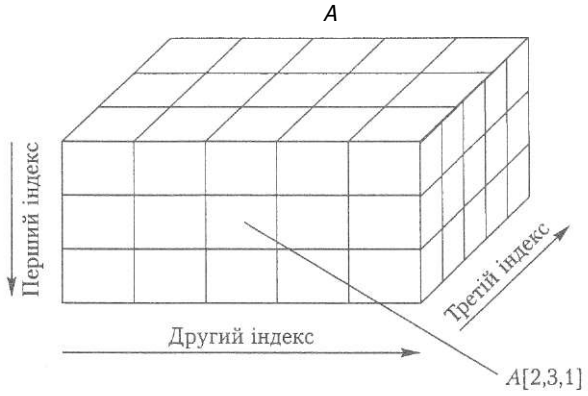


Рис. 7.17. Зображення тривимірного масиву

На рис. 7.18 подано зображення в оперативній пам'яті двовимірного масиву **a:array[1..4,1..4] of integer.**

«п	a 12	«13	«14	$a_2$1	a_2i	«23	«24	«31	«32	«33	«34	«41	Щ 2	«43	«44
	1-й рядок		I		2-й рядок		I		3-й рядок		'		4-й рядок		

Рис. 7.18. Зображення матриці в оперативній пам'яті

Такий спосіб зображення матриці в оперативній пам'яті дає можливість індексувати не тільки її елементи, але і цілі рядки. Наприклад, у випадку використання масиву **a:array[1. .4,1. .4] of integer** в результаті операції **a[1] :=a[3]** значення всіх елементів третього рядка будуть присвоєні відповідним елементам першого рядка. Водночас індексувати цілі стовпці матриці неможливо, що також пояснюється принципом збереження елементів матриці в оперативній пам'яті.

7.2.2. Базові операції обробки двовимірних масивів

Наведемо спочатку перелік базових операцій над матрицями та їх елементами. До таких операцій належать:

- введення та виведення матриць;
- створення нової матриці за заданим алгоритмом;
- пошук елементів матриці за певним критерієм;
- визначення, чи задовольняє матриця або окремі її елементи певній властивості;
- виконання певних операцій над компонентами матриць (переставлення рядків і стовпців, множення матриць тощо).

Тепер розглянемо типові алгоритмічні схеми обробки елементів матриці з метою подальшого застосування цих схем до розв'язання матричних задач. Зробимо загальне зауваження: однотипну обробку всіх елементів деякої матриці найлегше здійснити шляхом її обходу за рядками або стовпцями. Тобто спочатку обробляються всі елементи першого рядка (стовпця), потім — всі елементи другого рядка (стовпця) і т. д. Найпростішим методом реалізації такого обходу є використання вкладених циклів Tor. Отже, перейдемо до розв'язання базових матричних задач.

Введення матриці є достатньо очевидною операцією:

```
yar a:array[1..5. 1..5] oT iпїедег:
      i.o:iпїедег;
Ьеdіп
  Tor i:=1 їо 5 сіо          {зовнішнім цикл по рядках      }
    Tor з;=1 їо 5 сіо        {внутрішній цикл по стовпцях   }
      reacl(a[i ,Л)          {із клавіатури елементи
                              вводяться через символ пробілу }
епсі.
```

Стосовно виведення матриці слід зауважити, що у вікні результатів програми елементи матриці мають подаватись у вигляді таблиці. Тому переводити курсор на новий рядок слід лише після виведення всіх елементів поточного рядка:

```
yar a:array[1..5, 1..5] oT iпїедег;
    i,3:iпїедег;
Ьеді п
Tor i: =1 їо 5 сіо
  Ьеdіп
    Tor з;=1 їо 5 сіо
       мгіїе(a[i.3].' ');    {вивести елементи рядка          }
     мгіїеіп;                {перевести курсор на новий рядок}
   епсі;
епсі.
```

Аналогічну структуру має код, що ініціалізує всі елементи матриці деякими значеннями:

```
yar a;array[1..5, 1..5] oT iпїедег;
    i .3:iпїедег;
Ьеdіп
Tor i :=i їо 5 сіо
  Tor 3:=1 їо 5 сіо
    a[i.3];=10;    {альтернативи; a[i .3]:=ranclot(20):
                                  a[i,з]:=i*3: тощо}
епсі.
```

Одна з типових задач — пошук максимального або мінімального елемента матриці. Розв'язання цієї задачі нічим принципово не відрізняється від розв'язання аналогічної задачі для одновимірних масивів. У програмі ex7_13 здійснюється пошук значення та індексів максимального елемента матриці, яка ініціалізується

випадковими числами. Максимальний елемент запам'ятовується у змінній max, індекс рядка, в якому розташований цей елемент, — у змінній **i**max, а індекс стовпця — **у** змінній zmax.

Приклад 7.14

```
prodgat ex7_13;                {пошук максимального елемента    }
var a:array[1..5, 1..5] of integer;
    i,j:                       {індекси поточних елементів      }
    imax,jmax:integer;         {індекси максимального елемента  }
    max:integer;               {максимальне значення            }
begin
  randomize;
  for i:=1 to 5 do
    for j:=1 to 5 do
      a[i,j]:=random(20);
  max:=a[1,1];
  for i:=1 to 5 do
    for j:=1 to 5 do
      if max < a[i,j] then
        begin
          max:=a[i,j];         {запам'ятати максимальний елемент}
          imax:=i;             {та його індекси                 }
          jmax:=j;
        end;
end.
```

Потреба у перестановці або обміні значеннями між двома елементами багатовимірного масиву виникає, насамперед, у задачах упорядкування. У деяких задачах переставляють рядки або стовпці.

Перестановку рядків або стовпців можна виконати лише шляхом послідовних перестановок відповідних елементів. Наведемо фрагмент коду, що демонструє перестановку першого та п'ятого стовпців матриці. Нагадаємо, що для обміну значеннями між двома елементами слід використовувати допоміжну змінну.

```
var a:array[1..5, 1..5] of integer;
    i,                         {індекс поточних елементів у стовпцях }
    temp:integer;              {допоміжна змінна для виконання обміну}
begin
  for i:=1 to 5 do             {цикл по рядках                       }
  begin
    temp:=a[i,1];
    a[i,1]:=a[i,5];
    a[i,5]:=temp;
  end;
end.
```

Розглянемо такі операції, як вставка та видалення даних у багатовимірних масивах. Вставка та видалення окремих елементів матриці є, загалом, некоректними операціями, оскільки вони призводять до невизначеності щодо поведінки інших елементів: незрозуміло, слід зсувати елементи стовпця чи рядка, в яких відбувається вставка або видалення, чи здійснювати якийсь інший зсув. Тому найчастіше під час перетворення матриць операцію вставки або видалення поширюють на цілий рядок або стовпець. Але нагадаємо, що у будь-яких масивах будь-якої вимірності кількість елементів, яка була вказана під час оголошення, змінюватись не може. Тому вставка рядка або стовпця можлива тільки в межах оголошеної кількості рядків або стовпців. Видалення заданого рядка матриці демонструє програма ex7_14.

Приклад 7.15_

```
program ex7_14;                    {видалення заданого рядка матриці}
var a:array[1..5,1..5] of integer;
    i ,j,k,n,m:integer;
begin
  writeln('введіть КІЛЬКІСТЬ рядків та стовпців <= 5');
  readln(n,m);
  randomize;
  for i :=1 to n do
    for j :=1 to m do
      a[i, j]:=random(20);          {генерувати матрицю }
  writeln('Введіть номер рядка для видалення');
  readln(k);
  for i:=k to n-1 do                {видалити k-й рядок }
    for j:=1 to m do
      a[i ,j]:=a[i+1,j];
  n:=n-1;                           {зменшити кількість рядків}
end.
```

Цикл видалення *k*-*т о* рядка працює так: на місце *k*-*т о* рядка записують рядок *k* + 1, на місце (*k* + 1)-го рядка — рядок *k* + 2 і т. д., доки не буде вичерпано всі рядки. Оскільки обсяг пам'яті, що виділена під масив, під час його обробки не змінюється, то при видаленні будь-якого елемента масиву останній елемент буде дублюватися. Тому по завершенні циклу видалення слід зменшити на одиницю задану раніше кількість рядків.

7.2.3. Двовимірні масиви в задачах лінійної алгебри

У розділі 7.3.2 були розглянуті алгоритми обробки матриць. Наведемо декілька прикладів їх застосування з метою розв'язання задач лінійної алгебри. А саме побудуємо програмне розв'язання двох хрестоматійних задач: множення матриць та обчислення визначника квадратної матриці.

Множення матриць

Насамперед нагадаємо, що операція множення матриць $A \times B$ визначена тільки тоді, коли кількість стовпців матриці *A* дорівнює кількості рядків матриці *B*. Отже, *добутком* матриці *A* та матриці *B* називається матриця C, елемент якої c_y- дорівнює скалярному добутку г-го вектор-рядка матриці *A* та у-го вектор-стовпця матриці *B*. Це означення можна записати у вигляді рівності *C* = *AB*, де

$$\bullet a_i^T \times b_j = \sum_{k=1} a_{ik} \times b_{kj}, \quad = 1..Щ\ ] = 1../, к = 1..И.$$

Множення двох матриць можна зобразити такою схемою:

7' V ^ Ти «[б/

Лх *B* =

4 * 1

Тут *A* - (*m x* я)-матриця; *B* - (*nx* /)-матриця; *a]* - *i*-й вектор-рядок матриці *A*; *Ь*] - у'-й вектор-стовпець матриці *B*.

Приклад 7.16_

Наведемо програму, що обчислює добуток двох матриць. Розмірність і значення елементів матриць вводить користувач. У програмі передбачено обробку ситуації невідповідності матриць.

```
рrogram ex7_15:                              {множення матриць}
їуре Maїrix=array[1..10.1..10] oї reai:
yar i.,]:inїeдer:           {індекси елементів
    A,B:Maїrix;             {вхідні матриці
    C:Maїrix;               {матриця результату множення
    гом1,гом2:inїeдer;      {кількість рядків вхідних матриць
    coї1,co12:inїeдer:      {кількість стовпців вхідних матриць
{================== процедура введення матриці =========
рrocесиre inpuї(yar Ma$:Maїrix;yar line.koї:inїeдer);
{Ma$ - матриця, яку вводять; line.koї - КІЛЬКІСТЬ рядків і стовпців}
 Ьеgin
   гереаї
     мrїїeCinpuї пишbers oї гомз >=1  ');
     reaciin(iine):
     мrїїeCinpuї пишbers oї coїитпз >=1  ');
     reac11п(кo1);
     її (1ine<1) or (кo1<1) їїieп
                               мrїїe1п('ii''$ Teм, inpuї тоге'):
   иnїii (1ine>=1) anci (кo1>=1); {перевірка коректності даних}
   мrїїe1п('inpuї шaїrix:');
   Тог i :=1 їо line cio
```

```
      Tor з: =1 їо кої сіо
          reaciCMasC1.1]);
 епсі;
      ═══ процедура виведення матриці ═══
procesciиге оиїрии(Маз:Маїгіх;1іпе,ко1 :іпїедег);
{Маз - матриця:1 іПЕ,кої - КІЛЬКІСТЬ рядків і стовпців}
 Ьеділ
   Tor i: =1 їо line сіо
   Ьеділ
     Tor ,]:=! їо кої сіо
       мгіїе(Ma5[i.з]:6:2.    ');          {виведення елемента}
     мгіїеіп:                  {перевести курсор на новий рядок}
   епсі;
 епсі;
{═══════ множення матриць ═══════}
procesciиге глиіі;
 уаг к:іпїедег; {індекс елемента скалярного добутку векторів}
 Ьеділ
   іТ со11<>гом2 їііеп        {перевірка відповідності матриць}
     мгіїе!п('шиїїіріісаїіоп із ішроззіЬІе - шаїгіх are
              ипсопТогшаЬІе')
   еізе
     Ьеділ
       Tor і :=1 їо гомі сіо      {вибрати і-й вектор-рядок }
        Tor 1 :=1 їо соїі сіо     {вибрати 1-й вектор-стовпець}
        Ьеділ
          C[I.1]:=0;
          Tor к:=1 їо соїі сіо
            C[i.з];=C[i.o]+A[i.к]*B[кЛ;  {скалярне множення
                                          векторі в}
        епсі;
     епсі;
епсі;                                    {епсі оТ шиїі}
Ьеділ
  мгіїе!п('іприї шаїгіх А:  ');
  іприї(А,гом1,со11);                    {ввести матрицю А }
  мгіїе1п('іприї шаїгіх В;  ');
  іприї(В.гом2,со12);                    {ввести матрицю В }
  мгіїе1п('шаїгіх А ;');
  оиїрии(А.гомІ.соІІ);                   {вивести матрицю А}
  мгіїе!п( 'таїгіх В ;');
  оиїрии(В,гом2.со12);                   {вивести матрицю В}
  шиїі;                                  {перемножити А та В}
  мгіїе1п('гезиїі оТ шиїїіріісаїіоп :');
  оиїрии(С,гом1,со!2);                   {вивести матрицю С}
  геасііп;
епсі.
```

```
іприї шаїгіх Й:
іприї пипЬегз оГ гомз >=1  2
іприї пипЬегз о^ со1ипп5 >=1
іприї шаїгіх:
1 2 3
4 5 6
іприї шаїгіх В:
іприї пишЬег-5 о(= гои5 >=1  3
іприї пипЬег5 оїї со1ипп5 >=1
іприї шаїгіх:
2
2
2
шаїгіх й :
  1.«      2.00     3.00
  4Л       5.00     6. 00
шаїгіх В :
  2.00
  2.00
  2.00
гезиіі оїї тиііірііса ііоп
 12.00
 ЗВ.О0
```

РИС. 7.19. Результати роботи програми ex7_15. Множення матриць

Обчислення визначника квадратної матриці

Поняття визначника застосовне лише до квадратних матриць. Матриця називається квадратною, якщо кількість її рядків дорівнює кількості стовпців. Означення поняття визначника квадратної матриці можна знайти у будь-якому підручнику з лінійної алгебри (наприклад, [6]). Обчислення визначника за його означенням є неефективним, і на практиці цей метод не використовується. Нижче буде наведено програмну реалізацію набагато ефективнішого методу, що ґрунтується на зведенні матриці до трикутного вигляду. Перш ніж розглядати цей метод, дамо означення декількох базових понять.

Нагадаємо, що квадратна матриця має дві діагоналі. Діагональ, яка проходить від лівого верхнього кута матриці до її правого нижнього кута, називається *головною.* Елементи матриці, що мають рівні індекси рядків і стовпців, розташовані на головній діагоналі і називаються *діагональними.* Інша діагональ проходить із правого верхнього кута матриці до її лівого нижнього кута. Така діагональ називається *побічною.* Індекси елементів побічної діагоналі відповідають такій функціональній залежності: *} = n - i* + 1, де *i* — номер рядка, у — номер стовпця, а ге- розмірність матриці. Матриця називається *верхньою трикутною*, якщо значення всіх її елементів, що розташовані під головною діагоналлю, дорівнюють нулю. Визначник трикутної матриці дорівнює добутку всіх її діагональних елементів. Розглянемо алгоритм зведення матриці до трикутного вигляду.

На першому кроці алгоритму отримують нульові значення всіх елементів першого стовпця матриці, починаючи з другого елемента. Для цього визначаються

коефіцієнти виключення за формулою $к_{і} = -a_{ji}/a\backslash\backslash$ і до елементів г-го рядка додаються відповідні елементи першого рядка, помножені на $к_{і}$. На другому кроці алгоритму аналогічним чином отримують нульові значення всіх елементів другого стовпця матриці, починаючи з його третього елемента. Коефіцієнти виключення визначатимуться як $к_{і} = -0,2/022$ • На ці коефіцієнти множитимуться елементи другого рядка, і отримані результати додаватимуться до елементів інших рядків. Процес триватиме *n-1* кроків. Отже, значення елементів трикутної матриці визначаються за формулою

$$a< = a_{ц} \frac{-a_{IK} \times a_{K]}}{a_{KK}} A$$

Тут елемент *сім* — опорний елемент; *i,], к* - індекси рядків і стовпців (і = 1,2,..., *n, I* = 1, 2, ..., *n\к*= 1,2, ..., *n*); *n* - вимірність матриці.

Отже, на кожній ітерації алгоритму опорними є діагональні елементи. Це дільники, тому їх значення не повинні дорівнювати нулю. Якщо *г*-й опорний елемент дорівнює нулю, то *г*-й і будь-який інший рядок, в г-му стовпці якого міститься ненульовий елемент, треба поміняти місцями. Внаслідок такої перестановки рядків визначник змінює знак. Всі інші дії алгоритму не змінюють значення визначника.

Приклад 7.17_

Запишемо програму обчислення визначника квадратної матриці за поданим вище алгоритмом. Процедури іприї та оиірії виконують введення та виведення матриці, процедура Ігіапдиіаг(к:іпїедег) призначена для отримання нулів під головною діагоналлю в *к*-му стовпці, процедура сЬапде міняє місцями рядки, а процедура сіеіегтіпапі обчислює визначник.

```
prograt ex7_16;                    {обчислити визначник матриці  }
yar cieiireai;                     {значення визначника матриці  }
    tiniedeg;                      {вимірність матриці           }
    a: array[1. .5.1. .5]o1° rcai: {елементи матриці             }
    zidp:inPedeg:                  {ознака зміни знака визначника}
{============ введення елементів матриці ============}
procesiire iprii;
yar i,1:inIedeg;                    {параметри циклів            }
Ьедіп
  mrile!n('iприї 5l2€ oT шаїrix');
  reacПn(n);  {введення кількості рядків і стовпців матриці}
  mrile1n('iприї сотропеніз oT таїrix');
  Tor i :=1 їо n cio
    Tor 1: =1 їо n cio
    Ьедіп
      mrile('a[¹.i,'.',i.']=');
      reacI1n(a[i, Л);
    enci;
encsi;
{==========виведення значень елементів матриці ==========}
procesiire oиірии;
yar i.з:inїедег;                    {параметри циклів            }
```

```
Ьеділ
  Tor i :=1 їо n сіо
    Ьеділ
      Tor Д: =1 їо n сіо
        игіїе(a[i Л:5:2.'  '):
      мгіїеіп;
    епсі;
епсі;
{============ обчислення трикутної матриці ==================}
procesiire їгіапдиіаг(к:іпїедег);
                              {к - номер рядка, що обчислюється }
yar      іпїедег;             {параметри циклів                 }
    коеТ;rea1;                {коефіцієнт "виключення" елемента }
Ьеділ
  Tor 1: =к+1 їо n сіо        {вибрати наступний рядок          }
    Ьеділ
      коеТ;=a[1,к]/a[к,к];    {обчислити коефіцієнт "виключення"}
      Tor 3:=1 їо n сіо       {обчислити елементи у вибраному рядку}
        a[1,з]:=a[1.з]-a[к,з]*коеТ;
    епсі;
епсі;
і ================== перестановка рядків ====================}
procesiire cЬапде(і і: і пїедег):
{іі - номер рядка, що міняється місцем з будь-яким іншим рядком}
yar з,к:іпїедег;              {параметри циклів               }
    їглр: reai;     {тимчасова змінна для переставлення елементів}
Ьеділ
  5ідп:=1;    {ініціалізація ознаки зміни знака визначника}
Tor 3: =1 їо n-іі сіо          {пошук рядка з ненульовим опорним
                                елементом                     }
    іТ a[ii+3.ii]<>0 їііеп     {якщо шуканий елемент ненульовий, }
      Ьеділ
        Tor к:=1 їо n сіо      {рядки переставляються          }
          Ьеділ
            їшр;=a[ii,к];
            a[ii,к];=a[ii+з,к];
            a[ii+з,к]:=їтр;
          епсі;
          5ідп;=-5ідп;         {ознака зміни знака визначника  }
      епсі;
епсі;
;================== обчислити визначник ====================}
procesiire (іеїегшіпапї:
yar i:іпїедег;
Ьеділ
  Tor i :=1 їо n-1 сіо
    Ьеділ
      іТ a[i,i]=0 їЬеп      {якщо опорний елемент дорівнює нулю,}
        сііапде(і);          {переставити рядки місцями       }
      їгіапдиіаг(і);        {обчислити стовпець трикутної матриці}
    епсі;
```

```
  cleï:=1:
  Tor i :=1 їо n сіо          {перемножити діагональні елементи    }
    cleï:=cleï*a[i .i];
  iT зідп<0 їНеп йеї:=-сіеї;  {змінити знак визначника
ЄПС;
{=================головна програма ==============
Ьедіп
  игіїе1п('сіеїегтіпапї оТ шаїгіх');
  іприї;                      {ввести матрицю                }
  мгіїе!п('іпііїіаі шаїгіх');
  оиїриї;                     {вивести початкову матрицю     }
  сіеїегшіпапї;               {обчислити визначник           }
  «гіїе1п('їгіапдиіаг шаїгіх')
  оиїриї;                     {вивести трикутну матрицю      }
  мгіїе1п( 'с1еї=| ,сіеї:5:2); {вивести значення визначника}
  reacПп;
епсі.
```

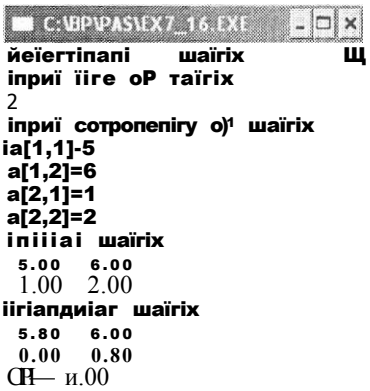


РИС. 7.20. Результати роботи програми ex7_16. Обчислення визначника квадратної матриці

7.3. Рядки

Один з різновидів одновимірних масивів - масив символів, або рядок, — посідає особливе місце у багатьох мовах програмування. І це не випадково, адже алгоритми перетворення рядків застосовуються для вирішення вкрай широкого кола задач: редагування та перекладу текстів, алгебричних перетворень формул, криптоаналізу, в інформаційно-пошукових системах, електронних словниках тощо. Більш того, відомо, що будь-який алгоритм можна подати у вигляді алгоритму перетворення рядків. У мові РазсаІ з усіх можливих типів масивів лише для рядків було зарезервовано стандартний структурований тип даних - зїгіпд, а також розроблено потужну бібліотеку процедур і функцій обробки даних рядкового типу.

7.3.1. Поняття рядка та оголошення змінних рядкового типу

Рядок — це скінченна послідовність символів, яку можна розглядати як особливу форму одновимірного масиву. Нагадаємо, що одна з характеристик масиву - це кількість його елементів, яка є фіксованою величиною і визначається під час оголошення масиву. Рядок як масив символів теж характеризується довжиною, тобто кількістю символів. Але для рядка розрізняють поняття *загальної* та *поточної* довжини. Загальна довжина рядка визначається об'ємом оперативної пам'яті, що була надана рядку під час його оголошення. Поточна довжина рядка визначається кількістю символів у ньому в конкретний момент виконання програми, вона ніколи не перевищує загальної довжини. Спосіб визначення поточної довжини рядка залежить від способу оголошення відповідної рядкової змінної. У мові РазcaI є два способи оголошення змінної-рядка:

- оголошення змінної спеціального структурованого типу даних зігіпд;
- оголошення змінної типу символьного масиву.

У разі оголошення рядка як змінної типу зігіпд його поточна довжина зберігається в елементі з нульовим індексом. У цей елемент записується символ, код якого дорівнює значенню довжини. При виведенні рядка користувач не побачить цього символу, але у програмі можна прочитати або змінити його значення. Але не слід забувати, що нульовий елемент рядка - це символ, а не число, і тому для отримання числового значення довжини рядка слід застосовувати вбудовану функцію orcI(<символ>), а для запису довжини - вбудовану функцію cHr(<число>).

У разі оголошення рядка як змінної типу символьного масиву його поточна довжина фіксується спеціальним символом, розташованим після останнього інформаційного символу рядка. Цей спеціальний символ називається символом кінця рядка або нуль-символом (>ТОЬЬ-символом), його А5СІІ-код дорівнює 0, а позначається цей символ лексемою #0. Зазначимо, що лише до тих рядків, які є змінними типу 5їгі пд, можуть бути застосовані бібліотечні функції обробки рядків, а також операції присвоєння, об'єднання та порівняння. Обробка символьних масивів є ніяк не легшою за обробку одновимірних масивів. Тому надалі розглядатимемо лише ті змінні-рядки, що оголошені як змінні типу зігіпд.

Зауважимо, що поточна довжина рядка визначається автоматично під час його ініціалізації. Зокрема, при введенні рядка з клавіатури його довжина дорівнюватиме кількості символів, введених до натискання клавіші **Enter.**

Настав час навести синтаксис оголошення змінної типу зігіпд:

<ім'я змінної>:зїгіпд[<загальна довжина рядка>];

Як бачимо, вказувати значення загальної довжини рядка не обов'язково. Отже, припустимим є скорочений синтаксис оголошення змінної рядкового типу:

<ім'я змінної>:зІгіпд;

Оскільки будь-який символ займає один байт пам'яті, то максимальне значення загальної довжини рядка буде обмежене максимальним значенням, яке можна записати в одному байті. Отже, максимальне значення довжини рядка становить

255 символів. Якщо під час оголошення рядкової змінної не вказана загальна довжина рядка, то за замовчуванням вона вважається рівною 255. Обсяг пам'яті, що виділяється для збереження значення рядкової змінної, буде на один байт більшим від вказаної в оголошенні загальної довжини рядка, оскільки для збереження значення його поточної довжини потрібний додатковий байт пам'яті.

Наведемо приклади оголошень змінних рядкового типу.

```
соnzі; 1епдУі=50:
уаг
   зіг:зігіпд;                    {резервується 256 байт}
   пате:зІгіпд[15]:               {резервується 16 байт }
   ас1сігез5:зігіпд[1епд1:Іі]:    {резервується 51 байт }
```

Як і елементи будь-якого масиву, символи рядка зберігаються у поряд розташованих комірках оперативної пам'яті. Припустимо, що до щойно оголошеної змінної **пате** було введено з клавіатури рядок символів **'Типьо Pascal'**. Зображення значення змінної **пате** в оперативній пам'яті буде таким, як показано на рис. 7.21.

♀		*и*	*г*	*Ь*	*о* I		*Р*	*а*	**5**	**с**	*а*	*І*			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Індекси елементів рядка

Рис. 7.21. Зображення рядка в оперативній пам'яті

В елементі з індексом 0 зберігається символ код якого дорівнює поточній довжині рядка **'Типьо Pascal'**, тобто 12. До елементів рядка з індексами від 13 до 15 можна звертатися: їм можна присвоїти значення будь-якого символу або прочитати їх. Але під час виведення рядка символи, що їх індекси перевищують поточну довжину, не відображатимуться.

Крім змінних рядкового типу використовуються *рядкові константи,* які записуються в одиночних лапках, наприклад: **'Типьо Pascal'**. Довжина рядкової константи визначається за допомогою функції **lепдІИ,** що її буде розглянуто в розділі 7.3.3. Цю саму функцію можна застосувати і для визначення довжини рядка, яка є значенням рядкової змінної.

7.3.2. Операції над рядками та рядкові вирази

У мові РазсаІ змінні типу **зігіпд** можна обробляти двома способами. Перший спосіб дає можливість розглядати рядок як цілісний об'єкт, другий — обробляти його як об'єкт, що складається з окремих символів, тобто з елементів типу **спаг.** Перший спосіб надає можливість за одну операцію присвоювати рядковій змінній значення рядка символів, виконувати об'єднання декількох рядків тощо. Другий спосіб забезпечує доступ до окремих символів рядка за їх індексами — аналогічно тому, як отримується доступ до елементів одновимірного масиву. Для доступу до символу рядка застосовують операцію індексування:

<ім'я змінної рядкового типу>[<індекс символу>]

Отже, розглянемо означені в мові РаscaI операції над даними рядкового типу. Це такі операції, як присвоєння, об'єднання та порівняння рядків.

Присвоєння рядків

Вираз рядкового типу можна присвоїти змінній рядкового типу. Символи рядка, що є значенням рядкового виразу, присвоюються відповідним елементам змінної. Якщо довжина рядкового виразу більше загальної довжини змінної, присвоюються лише перші символи в межах означеної кількості. Під час операції присвоєння неявно модифікується нульовий елемент змінної, що набуває нового значення. Нехай змінна **зіг** означена як **зїгіпд[5].** Після присвоєння **51г: =' KiyV** значення символів рядка стануть такими: **зIr[0]=cьr(4), зїг[1]='K', зїг[2]='i', зIr[3]='y', 5'Ъг[4]=' V**'. Якщо згодом виконати оператор присвоєння **зіг: ='Ilkraine**', то символи '**пе**' в рядку **зіг** будуть втрачені.

Об'єднання рядків

Над рядковими змінними означена операція *об'єднання,* або *конкатенації,* яка дозволяє дописати один рядок в кінець іншого. Ця операція позначається символом «+». Під час конкатенації рядків поточна довжина рядка, до якого приєднується інший рядок, збільшується на довжину рядка, що додається. Збільшення довжини об'єднаного рядка здійснюється тільки у заданих під час оголошення межах. Наступний приклад демонструє конкатенацію рядків та ситуацію, коли така операція виконується некоректно через накладені на загальну довжину рядка обмеження.

Приклад 7.18__

Ввести рядки, об'єднати їх та вивести результат.

```
Рrодгаш ex7_17;
уаг
    пате: з1:гі пд[5];
    асicirез5:з1гіпд[12]:
    5Pr1:зргі пд[7];
    зPr2:5Prinд;
    i cienPI 'T: 5l;ri пд[15]:
Ьеділ
    мгіїеIпС'епїег  пате');
    геасПп(пате);
    игіЬеIпСIепдIН  оїї  пате=' ,orc1(пате[0]));
    мгіїе"Iп('епїег  асicirезз');
    геасЯп(асIcIre55);
    «rїieInC 'IепдіН  оТ  асicirезз=' .orcKacIcirеззСО]);
    зїг1:=паше+'  'ча<1cire55:
    мгіїеin(зїг1);
    «riPe1n(  Чепдіь  оїї  зїг1=',orcI(5Їr1[0])):
    зPr2:=пате+'  'ьасicirезз;
    «rііе1п(5їг2):
    мгіїеInC IепдіН  оТ  зїг2='.orci(5їr2[0])):
```

```
    icient1iT:=name+' '+aci(ire$$;
    мгїieIn(icIeniiT);
    мгile1n(' IепдIli oT 1cienIiT=',orci(ic1eniiT[0])):
    reaciin;
епсі.
```

На рис. 7.22 зображені результати виконання програми **ex7_17.** З цього рисунка видно, що під час об'єднання рядків завдовжки 3, 1 та 12 символів у рядок, для якого виділено тільки 7 байт пам'яті **(зігі :=па<пе+' '+acMre35;),** праві символи рядка асісігезз втрачаються. Під час конкатенації рядків завдовжки 3 та 12 символів у рядок завдовжки 255 символів **(5Ir2:=пате+' '+acMre5з;)** немає жодних втрат.

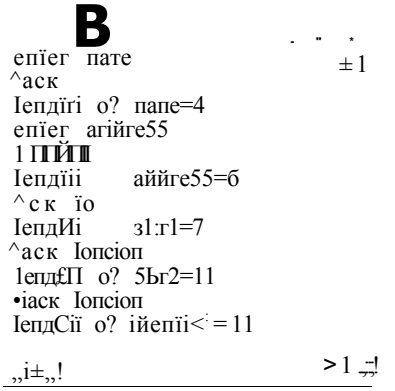


Рис. 7.22. Результати роботи програми ex7_17. Конкатенація рядків

Порівняння рядків

Для даних рядкових типів означено всі наявні в мові RascaI операції порівняння: =, <>, <, >, <=, >=. Щоб зрозуміти принцип дії цих операцій, нагадаємо спочатку правило порівняння даних символьного типу. Кожному символу відповідає унікальний код у таблиці символів A5CII. З двох символів більшим вважається той, код якого більший. Рядки — це послідовності символів, і порівняння рядків виконується як серія порівнянь тих символів рядків, що мають однакові індекси. Інакше кажучи, при порівнянні рядків враховується їх *лексикографічний,* або *алфавітний,* порядок. А саме, спочатку порівнюються перші символи рядків (символи з індексами 1). Якщо ці символи відмінні, то більшим вважається той рядок, перший символ якого більший. В іншому разі порівнюються другі символи рядків. Якщо і другі символи рядків виявляються однаковими, то порівнюються треті символи рядків тощо. Отже, якщо а і Ь - деякі рядки, а і - цс такий найменший індекс, що а[і] <> **Ь[і],** то рядок **а** вважається більшим у тому разі, коли **а[і 3 > Ь[і],** і навпаки. Постає питання: а коли такого індекса **і,** що **а[і] <> Ь[і**], не існує? Тоді можливі два варіанти: коли довжина рядків однакова, то рядки вважаються рівними, а коли довжина рядків різна, то більшим вважається довший рядок. Для прикладу наведемо декілька істинних булевих тверджень.

```
'Borlancl' < 'Turbo'
'Borlanci Pascal' < 'Turbo'
'Turbo Pascal' > 'Turbo'
'Borlanci' > 'ВОГІЗОУ'
'Borlancl' < 'borlancl'
'borlancl' < 'борланд'
```

Істинність цих тверджень випливає зокрема з того, що A5CII-коди великих літер менші за A3CII-коди маленьких літер, коди літер латиниці менші за коди літер кирилиці, а літери того самого регістру з тієї самої абетки розташовані в таблиці A5CII-K0ДІВ у алфавітному порядку.

7.3.3. Процедури та функції обробки рядків

Потужна бібліотека процедур і функцій, призначених для роботи з рядками в мові PascaI, значно полегшує розв'язання задач, пов'язаних з обробкою текстів. У цьому розділі розглянуто синтаксис і семантику рядкових процедур і функцій, а також розв'язано за їх допомогою практичну задачу. Одразу зазначимо, що для введення та виведення рядків застосовують стандартні процедури введення-виведення даних. Наприклад, для рядкової змінної **з** можна виконувати процедури **reacПn(з) тамгііеіп(з).** Використання процедур введення та виведення рядків було проілюстровано в прикладі 7.18. У даному розділі ці процедури не розглядаються.

Процедури обробки рядків

Процедура видалення дає можливість видалити певну кількість символів з рядка, починаючи із заданої позиції. Її прототип такий:

```
0e1ele(yar з: 3irinд; Inciex: Inїедег; Coini:Inїедег);
```

Вказана процедура має три параметри: з — рядок, з якого видаляються символи; **Inciex** - індекс символа, з якого починається видалення; **Coini** - кількість символів, що мають бути видалені.

Під час видалення довжина рядка автоматично зменшується на кількість видалених символів. Символи, що містилися в рядку праворуч від зони видалення, автоматично переставляються на позиції видалених символів, тобто здійснюється серія операцій присвоєння з[і]:=з[і+с], де з — змінна рядкового типу; с - кількість символів, що видаляються; і — лічильник. Наведена нижче програма видаляє з рядка 'сііїїсігеп' 3 останніх символи.

```
yar з1r:з1rinд;
Ьеgiп
    3ir:='ciii1ciren';
    Oelele(зIr.6.3):
    Kriiein(зir);        {виведено 'СЬIICI'}
епсі.
```

Зворотна операція вставки символів у рядок реалізується процедурою **Inзerі,** що її параметрами є два рядки й одне ціле число:

```
Inзerl(5оигсе: 3irinд; yar з: 3irinд; Inciex: Inїедег);
```

Зміст параметрів процедури є таким: 5оигсє — рядок, що вставляється в інший рядок; з - рядок, куди здійснюється вставка; Іпсіех - номер позиції, з якої вставка починається.

Після виконання вставки символів довжина рядка автоматично збільшується в межах заданої в оглошенні довжини. Якщо вставка здійснюється всередину рядка, то місце для підрядка вивільняється за рахунок зсуву символів рядка вправо, тобто виконується серія присвоєнь вигляду з[i+c]:=з[i]. Як приклад наведемо програму, що вставляє у рядок 'ТпсГ символи 'іє'.

```
yar 5Ir1,зIr2:5Iппд;
Ьеділ
  Біті:='їгпсі';
  $їг2:='ie':
  i пзerKзIr?,5Ir1.3);
  мгіРеї псБргі);              {виведено 'Tr і епсі'}
епсі.
```

Рядки, що складаються із символів цифр, можна перетворити на числа і навпаки. Для цього використовують процедури **Уаі** і 51г, прототипи яких наведено нижче.

```
Va 1 (з: 21гіпд; yar y; yar Cocie: Іпїедег);
```

Процедура має такі параметри: з - рядок, що перетворюється на число; V - змінна типу **іпїедег** або **reai,** до якої число буде записано; Cocie - змінна цілочислового типу, яка визначає успішність перетворення (Cocie = 0) або містить код помилки **(Cocie<>0).** Помилка виникає тоді, коли рядок 5 не є символьним записом числа. У такому разі значення змінної Cocie дорівнює номеру першого помилкового символу. Як приклад наведемо програму, що перетворює рядок на значення цілочислової змінної Мит.

```
сопзі зІ:='150';
yar піт, cocie: і пїедег;
Ьеді п
  Уаі (зі,пит,cocie);
  мгїіеіп(пигп):              {виведено 150}
епсі.
```

Процедура 51г є зворотною до функції Уаі. Наведемо її прототип:

```
51r(x [ І^icШі [: Oecітаїз ]]; yar з:з1гіпд);
```

У процедурі означено такі параметри: х - змінна цілого або дійсного типу, значення якої перетворюватиметься на рядок; Мі сіІН - необов'язковий параметр, що визначає кількість символів у створюваному рядку; Ресітаїз — необов'язковий параметр, що визначає кількість символів після десяткової точки; з — рядок, який утворюється. Наведемо код, що перетворює число на рядок за допомогою функції 51.

```
yar N:reai;
    5I:зIгіпд;
```

```
Ьеділ
  п:=160.236;
  зІг(п:5:2,зІ):
  игііеіп(зі);      {виведено 160.24}
епсі.
```

Функції обробки рядків

За допомогою бібліотечних функцій, що мають параметри типу зігіпд, можна визначити поточну довжину рядка, скопіювати рядки та їх підрядки, визначити позицію входження підрядка в рядок та виконати конкатенацію рядків.

Отже, наведемо прототип функції копіювання підрядка в новий рядок:

```
Copy(з: Зігіпд; Іпсіех: Іпїедег; Соипі: Іпїедег): Зігіпд;
```

Параметри функції: з - рядок, який копіюють; Іпсіех - позиція, з якої розпочинається копіювання; Соипі — кількість символів, що копіюються. Якщо загальна довжина рядка, який утворюється, є меншою за кількість символів, що копіюються, то у вихідному рядку будуть тільки перші символи з числа тих, які треба скопіювати. Наведемо приклад:

```
уаг з: зігіпд;
Ьеділ
  з:='ТигЬо Pascal';
  з:=Copy(з, 7. 6);  {результат - рядок символів 'Pascal'}
епсі.
```

Для визначення поточної довжини рядка з крім операції огсі(5[0]) можна застосувати функцію І-епдІіі(з). Її прототип є очевидним:

```
ІепдЩз: зігіпд): Іпїедег:
```

Операції лексичного аналізу потребують визначення наявності певних символів у рядках. У мові Pascal означена функція, що повертає номер позиції першого входження символу або підрядка в рядок. Прототип функції такий:

```
Роз(5иЬзІг: зігіпд; з: зігіпд): Ьуіе;
```

У функції означено два параметри: ЗиЬзІг — підрядок, позицію першого входження якого необхідно повернути; з — рядок, до складу якого має входити підрядок. Якщо потрібного підрядка в рядку немає, то функція повертає нуль. Наведена нижче програма всі пробіли на початку рядка з замінює цифрами 0.

```
уаг з; зігіпд;
Ьеділ
  3:='    123.5';
  міііе Роз(' '. з)>0 сіо
    з[Роз(' '. з)]:='0';
епсі.
```

Наведемо приклад застосування процедур та функцій обробки рядків.

Приклад 7.19__

Будь-який текстовий редактор може виконати операцію заміни у тексті всіх входжень одного рядка іншим. Поставимо задачу замінити всі слова 'сілісі' у введеному користувачем рядку словом 'сіїі І сіген', розуміючи під словом деяку послідовність символів, не обов'язково оточену пробілами. Зауважимо, що наведена нижче програма ex7_18 розв'язує дану задачу не найоптимальнішим способом, вона призначена для демонстрації можливостей якомога більшої кількості бібліотечних процедур і функцій.

Алгоритм заміни в рядку кожного слова 'сЬіїсі' словом 'сЬіїсіген'

1. Ввести рядок.
2. Створити новий порожній рядок
3. Поки у вхідному рядку міститься підрядок ' сіїі ІсГ, повторювати дії, зазначені у пунктах 3.1-3.4.
   3.1. Визначити позицію поточного слова 'сНіїсГ.
   3.2. Вставити підрядок ' геп' у вхідний рядок на визначену позицію.
   3.3. Приєднати до нового рядка копію вхідного рядка, починаючи з першого символа до слова 'сИ 1 сіген' включно.
   3.4. Видалити із вхідного рядка скопійований підрядок
4. Після завершення циклу приєднати до нового рядка символи вхідного рядка, що залишилися після видалення.

```
Ргодгат ex7_18:                          {заміна слів у тексті}
 уаг з:зРгіпд;                              {початковий рядок}
     п:і пїедег;                            {позиція в рядку }
     5пеи:$1гіпд;                           {новий рядок     }
 Ьеді п
   мгіїеInCTo гepiace "сіїі 1 сі" иіїН "cMIcIreп" ');
   мгіїеіп ('іприї зїгіпд');
   reaciin(з);
   зпем:='';          {порожній рядок, до якого додаватимуться
                       підрядки зі вставленими символами      }
   мНіie Роз( 'cНіїсГ. з )<>0 сіо
    Ьеділ             {поки в рядку трапляються символи 'cНіїсГ}
                     (визначати позицію поточного слова 'cНіїсГ }
     п;=Роз( 'cЬіїсі' .з):
                     (вставити 'геп' у рядок після слова 'cНіїсГ}
     іпзегї('геп',$,п+5);
                                {до нового рядка додати частину
                                 рядка з. від його початку
                                 до слова 'смісіген'          }
     зпеи:=зпем+Copy(з, 1,п+5);
                     {видалити з рядка з скопійований підрядок}
     cIe1eїe(з,1,п+5);
    епсі;
```

```
  знем:=5пем+з; {приєднати рядок після останнього 'сИilri'}
  мri1e1n('оиїриї зїгіпд');
  мгіїеin(знем);
encl.
```

```
I - P I XI
To repiace "CПIIЙ" їо "cHilйren"
inриї зїгіпд
сГііїй дое5 їо 5Clioo1. СҐШЙ eaї5 appie.
оиїриї зїгіпд
ciiiїйгеп дое5 їо 5Cґioo1. СПШЙСЄП eaЇ5 appie.
```

РИС. 7.23. Результати роботи програми ex7_18. Заміна слова 'сПііїй' словом 'cHiiciren'

Висновки

- Одновимірний масив — це послідовність однотипних даних. Масив поєднує в собі дві структури: множину елементів і заданий на цій множині порядок.
- Тип елементів масиву називається базовим типом.
- Порядок у масиві визначається набором значень одного й того самого типу, що називається індексним, а самі ці значення називаються індексами. Індексний тип має бути простим порядковим типом даних.
- Кількість елементів в одновимірному масиві становить його розмірність, або довжину. Ця кількість є фіксованою, задається в оголошенні масиву та не може змінюватися під час виконання програми.
- Усі елементи одновимірного масиву записуються до розташованих поряд комірок оперативної пам'яті.
- Двома найпростішими операціями над елементами одновимірного масиву є вибір деякого елемента та зміна його значення. Для доступу до окремого елемента масиву використовується операція індексування [], а для зміни його значення - операція присвоєння.
- Тип масиву — це структурований тип даних, множина допустимих значень якого складається з усіх масивів, для яких зафіксовано розмірність, базовий тип, індексний тип і множину значень індексу.
- Обробка масивів полягає в послідовному виконанні операцій над їх елементами. Для масивів як цілісних об'єктів означено лише операцію присвоєння.
- Масиви можна використовувати як параметри процедур і функцій. Масиви можуть передаватися до підпрограм як параметри-значення або як параметри-змінні. При передачі масиву як параметра-значення утворюється його копія, а при передачі масиву як параметра-змінної передається лише адреса його першого елемента.

- Двовимірний масив, або матриця, - це масив, кожний елемент якого є масивом. Елемент матриці має два індекси, що ними визначається номер рядка та номер стовпця.
- Рядок - це скінченна послідовність символів, що її можна розглядати як різновид одновимірного масиву. У мові РазсаІ означено спеціальний рядковий тип **зігіпд.**
- Для рядка означено поняття загальної та поточної довжини. Загальна довжина рядка дорівнює обсягу оперативної пам'яті, що була надана рядку під час його оголошення. Поточна довжина рядка визначається кількістю символів у рядку в конкретний момент виконання програми.
- Максимальна загальна довжина рядка становить 256 символів.
- У мові РазсаІ змінні рядкового типу можна обробляти двома способами. Перший спосіб дає можливість розглядати рядок як цілісний об'єкт, другий — як об'єкт, що складається з окремих символів, тобто з елементів типу сііаг.

Контрольні запитання та завдання

1. Дати означення масиву та типу масиву.
2. Якими є головні властивості масивів даних?
3. Яким є принцип зображення двовимірного масиву в оперативній пам'яті?
4. Як здійснюється доступ до елементів одновимірного масиву та матриці?
5. У чому полягає відмінність між передачею масиву до підпрограми як параметра-значення та як параметра-змінної?
6. Чи може список ініціалізації масиву містити більше (менше) значень, ніж вказано в оголошенні масиву?
7. Чи можна, не використовуючи циклів, поміняти місцями рядки (стовпці) матриці?
8. Чим відрізняється поточна довжина рядка від його загальної довжини?
9. Де в оперативній пам'яті зберігається поточна довжина рядка?
10. Як ініціалізувати рядок під час його оголошення?
11. Як увести та вивести рядок?
12. Які бібліотечні процедури та функції визначені для змінних рядкового типу?

Вправи

1. У твердженнях заповнити пропуски.
   1.1. Елементи масиву мають один і той самий_____.
   1.2. Значення, що використовується для доступу до елементів масиву, називається_____.

1.3. Під час роботи з масивом_____його елементів не змінюється.

1.4. Масив, елементами якого є одновимірні масиви, називається

1.5. Загальна довжина рядка не може перевищувати_____символів.

2. Оголошено такі масиви:

```
a:array[1..15,0..3] of cHar;
Ь:array[1.15] of array[0..3] of cïïar;
```

Чи однакові типи масивів **a** та **Ь?**

3. Оголошено такі змінні:

```
a.Ь:array [1..15,0..8] of reai;
ï: Ьooïean;
```

Визначте коректні операції:

```
a :=Ь;
a:=a+Ь;
ï:=a<>Ь;
Keacl(a);
a[1]:=Ь[15];
a[2,4]:=Ь[1,2]+Ь[5,3]:
```

4. Наведена нижче програма має копіювати елементи масиву x до масиву y. Виправити помилки у програмі.

```
yar x:array [1..40] of cьar;
    y:array [0. .39] of cïïar;
    i:inïeдer;
Ьeдin
   {введення масивів}
   їor i :=0 їo 40 cio y[i]:=x[i];
епсі.
```

5. Які з наведених нижче операторів присвоюють цілочисловій змінній x значення довжини рядка 5?

```
x:=orcl(5[0]);
x:=1епдїН(з);
x:=OГC!(5[256]);
x:=5;
```

6. Визначити, що буде виведено на екран у результаті виконання наведеної нижче програми (повідомлення про помилку, символи 1234 чи символи 12):

```
yar 5ї:5їrіпд[2];
Ьeдin
   5Ї:='1234';
   мгіїе1п(5ї);
епсі.
```

Задачі

1. Згенерувати значення елементів одновимірного масиву за допомогою генератора псевдовипадкових чисел, ввівши кількість елементів масиву з клавіатури. Знайти мінімальний за значенням елемент і записати його на початок масиву, вивільнивши для нього місце шляхом зсуву елементів масиву вправо.
2. Ввести значення елементів одновимірного масиву, задавши попередньо їх кількість. Визначити кількість входжень до масиву значення кожного з його елементів.
3. В одновимірному масиві обчислити кількість елементів у найдовшій серії. Серія — це послідовність однакових елементів, розташованих поряд.
4. Згенерувати додатні та від'ємні псевдовипадкові значення елементів одновимірного масиву, ввівши кількість елементів з клавіатури. Переставити елементи масиву так, щоб спочатку були розташовані всі додатні елементи, потім всі від'ємні. Порядок серед додатних і від'ємних елементів має зберегтися.
5. Задати матрицю розмірності *mx n (m,n>* 3) Починаючи з лівого нижнього кута матриці та рухаючись лише праворуч і догори, досягти її правого верхнього кута і вибрати при цьому такі значення елементів, що їх сума буде максимальною. Вивести перелік вибраних елементів.
6. Знайти добуток довільної кількості матриць довільної розмірності. Кількість матриць, які треба перемножити, їх розмірність і вміст вводяться з клавіатури.
7. Межа саду має форму багатокутника з *n* вершинами, координати яких *(x\, y{),* (x_2. *Уг)>* •••> *(xn > Уп)* користувач вводить з клавіатури. Садівник вирішив обійти сад уздовж межі в порядку зростання номерів вершин багатокутника. З'ясувати, залишатиметься сад при цьому зліва від садівника чи справа.
8. У заданій квадратній матриці знайти суму значень елементів, що розташовані на головній діагоналі або вище від неї і є більшими за всі елементи, розташовані нижче від головної діагоналі.
9. У заданій квадратній матриці значення деяких діагональних елементів дорівнюють нулю. Переставити рядки або стовпці матриці таким чином, щоб діагональні елементи стали ненульовими. Якщо це неможливо зробити, вивести відповідне повідомлення.
10. Задати квадратну матрицю, ввівши кількість рядків і стовпців з клавіатури. Упорядкувати значення елементів головної діагоналі за алгоритмом вставки, а значення побічної діагоналі - за алгоритмом обміну. Визначити кількість порівнянь та обмінів під час сортування.
11. У квітниковій крамниці є *P* букетів квітів, пронумерованих від 1 до *P,* та У ваз (У > *P),* які розташовані на полиці і пронумеровані від 1 до *V.* Букети можна ставити до різних ваз, по одному до кожної, дотримуючись такого правила: якщо *i <],* де *i* та; — номери букетів, то букет *i* повинен міститися у вазі, що стоїть лівіше від вази з букетом}. Розташуванню певного букета у певній вазі приписується деяка естетична характеристика, що позначається натуральним числом. Естетична характеристика порожньої вази дорівнює нулю. Треба визна-

чити спосіб розподілу букетів по вазах, який максимізував би суму значень естетичних характеристик. Результат подати у вигляді одновимірного масиву, ***к***-й елемент котрого дорівнює номеру вази, у якій розміщено ***к***-й букет.

12. Здійснити обхід матриці по спіралі за годинниковою стрілкою, починаючи від її лівого верхнього кута. Вивести елементи матриці у порядку їх обходу.
13. Шахова фігура «кінь» переміщується на 1 клітину по горизонталі і на 2 клітини по вертикалі або на 2 клітини по горизонталі і на 1 - по вертикалі. «Кінь» розпочинає свій шлях з нижнього лівого кута і переміщується по шаховій дошці, яка має *пх п* клітин. Визначити кількість способів, якими «кінь» може дійти до правого верхнього кута дошки за ***к*** кроків.
14. У селищі, де *N* будинків, розташованих уздовж прямої дороги з однієї сторони на рівних відстанях, прокладають телефонний зв'язок. Зазначено, скільки телефонних апаратів треба встановити в кожному будинку. Кожен телефон має бути з'єднаний з АТС окремим кабелем. Визначити, в якому будинку необхідно встановити АТС, щоб сумарна довжина кабелів була мінімальною.
15. Ввести рядок символів, у якому містяться круглі дужки. Перевірити, чи є баланс дужок у рядку: для кожної дужки, яка відкривається, справа має бути дужка, що закривається. Дужки можуть бути вкладені одна в одну.
16. У рядку символів знайти найдовше слово. Словом вважатимемо послідовність символів, що відокремлена від інших символів довільною кількістю пробілів та не містить пробілів усередині.
17. З рядка символів видалити слова, номери яких парні. Серед слів з непарними номерами визначити найдовше.
18. Ввести два рядки символів, Видалити з другого рядка ті слова, що є у першому рядку.
19. Видалити з рядка символів зайві пробіли, залишивши по одному пробілу між словами.
20. Ввести рядок, перетворити послідовності цифрових символів в ньому на числа та знайти їх суму.
21. У рядку символів визначити кількість повторень кожного слова та видалити дублікати слів. Слова відокремлюються пробілами.
22. Заданий рядок Після видалення однієї літери з 5] утворюється рядок 5_2. Після видалення з 5! іншої літери утворюється рядок 5_3. Потрібно за рядками 5_2 і 5_3 відновити рядок

Розділ 8

Записи та **множини**

- Поняття запису
- Операції над записами та їх компонентами
- Способи роботи з масивами записів
- Записи з варіантами
- Поняття множини в мові Pascal
- Операції над множинами
- Зображення множин в оперативній пам'яті
- Приклади програм, у яких використовуються записи або множини

8.1. Записи

У попередньому розділі було розглянуто одну з найважливіших структур даних - масив. Визначальною характеристикою масиву є однорідність, тобто однотипність його елементів. Проте реальний світ насичений неоднорідними структурами даних. Прикладами таких структур можуть стати: календарна дата, що складається з номера дня, номера року та назви місяця, особова картка працівника (прізвище, адреса, дата народження, прибуток тощо), відомості про товар (назва товару, ціна, виробник, номер сертифіката якості тощо). Зрозуміло, що масив не є зручним та адекватним засобом зберігання структур різнотипних даних. Адже для зображення, скажімо, множини календарних дат довелося б створити три масиви: цілочислові масиви років і номерів днів, а також рядковий масив назв місяців. Такий спосіб зберігання даних є потенційно небезпечним, оскільки, зокрема, не виключає можливості зсуву в масиві місяців без відповідного зсуву в масиві днів. Отже, обробка згаданих масивів має проводитись синхронно і тому є достатньо складною задачею. Набагато зручніше було б створити один масив, що зберігатиме календарні дати як структури. Кожна із структур міститиме дані різних типів, але при цьому всі структури належатимуть тому самому структурованому типу даних. Структури, що можуть об'єднувати дані різних типів, в мові Разсаі називаються записами. Записам і присвячено розділ 8.1.

8.1.1, Запис та його оголошення

Запис — це структурований тип даних, що являє собою об'єднання фіксованої кількості змінних одного або декількох типів. Змінні, що входять до складу запи-

су, називаються його *полями.* Можливість інтегрування в один запис різнотипних компонентів становить головну відмінність запису від масиву. Зауважимо, **що,** на відміну від терміна «масив», терміном «запис» позначатимемо саме тип даних, а дані цього типу називатимемо екземплярами записів. Таке застосування термінології пояснюється тим, що для типу запису, як правило, оголошується окремий ідентифікатор, а тип масиву найчастіше оголошується неявно, разом із **оголошенням** змінної масиву.

Оголошення записів як типів даних здійснюють в розділі їуре Разсаі-програми. Після цього їх можна використовувати для оголошення змінних типу запису - це робиться в розділі оголошення змінних уаг. Оголошення запису та змінної типу запису має такий вигляд:

```
їуре <ім'я типу> = recor<1
                    <ім'я поля1>:<тип>;

                    <ім'я поляМ>:<тип>;
                  ЄПСІ:
уаг <ім'я змінної>:<ім'я типу>;
```

Оголошення типу розпочинається ключовим словом recorcsi і завершується словом епсі. Між цими словами міститься список оголошень полів. Кожне поле оголошується як ідентифікатор змінної певного типу. Зазначимо, що ім'я поля має бути унікальним в межах запису. Наведемо приклади оголошень записів і змінних відповідних типів.

Приклад 8.1

```
їуре
сіаїе=recorci
        йау:1..31:
        топІМ:5ЇГІ пд;
        уеаг:1..2010:
      епсі;
регзоп=recorcl
          Ті гз1:_пате: зігі пд:
          1а$1_пате:$£ппд;
          ac!clre55:5'Ьгіпд;
          Ьі гШау: сМе;
          proTii: reai:
        епсі:
ргосіисе=recorc1
           пате:вігіпд:
           price:reai;
           ргосіисегізїппд;
           сегїі Ті саіі оп:і піедег:
         епсі:
уаг сІ:(Ме;
    тап:регзоп:
    іТет: ргосіисе:
```

Поле запису може мати будь-який тип, крім файлового. Зокрема воно може бути змінною структурованого типу. Обсяг пам'яті, що виділяється транслятором для зберігання даних типу запис, визначається як сума обсягів областей пам'яті, що виділені для зберігання всіх його полів. Так, для зберігання даних типу ciaie з наведеного вище прикладу потрібно 259 байт пам'яті (1+256+2), а даних типу регзоп - 1033 байт (256 • 3 + 259 + 6).

Записи можна використовувати і для створення типізованих констант. Наприклад, константа УісїогуОау оголошеного у прикладі 8.1 типу ciaie може мати такий вигляд:

```
сопзі VIcIoryOay:ciaie = (ciay:9: топІН:'тау'; year: 1945):
```

8.1,2, Доступ до компонентів та операції над записами

У розділі 8.1.1 згадувалося, що під час оголошення запису кожен його компонент позначається ідентифікатором, який має бути унікальним у межах запису. Але слід зауважити, що імена полів різних записів можуть збігатися. Тому звернення до компонентів записів здійснюється через *складене ім'я,* що має такий синтаксис:

<ім'я змінної типу запису>.<ім'я компонента>

Крапку, що записується між іменем запису та іменем компонента, інколи називають символом оператора доступу. Якщо екземпляр деякого запису є полем іншого запису, то складені імена полів такого екземпляра міститимуть декілька символів оператора доступу. У прикладі 8.1 розглядається запис регзоп (людина), поле ЬігШау (день народження) котрого є екземпляром іншого запису - ciaie. У такій структурі ідентифікатор тап.Ьі гШау.yeaг адресує рік народження людини тап. Складність доступу до компонентів записів може бути зменшена, якщо використати конструкцію мі ІИ, яку ще називають оператором приєднання. Синтаксис конструкції мі їй такий:

```
иіІН <список змінних типу запис> cio
Ьеділ
  <оператори>
епсі;
```

В операторах всередині конструкції мі ІН замість імен змінних типу запис вказуються лише імена полів. Дія однієї конструкції міііі розповсюджується на вміст лише одного складеного оператора і лише на ті записи, імена яких були вказані у списку змінних типу запис.

Приклад 8.2

Нехай оголошено запис регзоп (ім'я, адреса, день народження, прибуток), у якому компонент «адреса» сам має тип запису Ноте_аclclre55 із компонентами «місто», «вулиця», «номер будинку». Схематично структуру екземпляра тап запису регзоп зображено на рис. 8.1, а нижче наведено програму, що ініціалізує поля ТігзІ_пате та асісігезз цього запису.

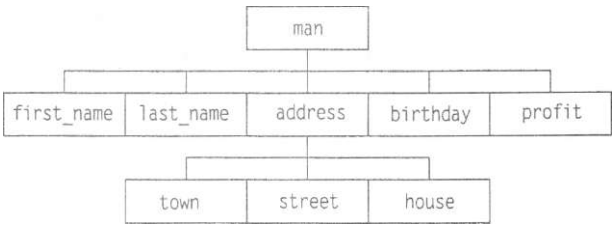


Рис. 8.1. Структура запису

```
Iype
  Иоше_acIc1re55=recorcl
    Io\л/п: 5Iгiпд;
    5їгєе'Ь:5'Ьгіпд;
    Ноизе:Ьуle;
  епсі;
  регзоп=recorci
    TiГ5i_паше:зігіпд;
    1азї_паше:5I:гіпд;
    acicігезз: Иоше_acIcIгезз;
    ЬігIIic1ay:$Iгіпд;
    proTiI:rea1;
  епсі;
уаг тап;регзоп;
Ьеділ
  «Піл тап.acicігезз сіо
  Ьеділ
    TігзI_пате:='Iуап';
    ІОМІ: = 'КІЄУ '; зігееї:='Уісїогу'; Іюизе:=42;
  епсі;
епсі.
```

Якщо ім'я відповідної структурної змінної в конструкції міІН не вказано, дія цієї конструкції не поширюється на поля даного запису, наприклад,

```
міііі тап сіо
Ьеділ
  TігзI_пате:=' Іуапоу';
  acicігезз.£ОШ: = 'КІЄУ' ;
  acicігезз. 5"Ьгее1:: ='VIciory';
  acicігезз .Ноизе; =42;
епсі;
```

Над компонентами записів можна здійснювати будь-які операції, що є допустимими для типів цих компонентів. Для екземплярів запису як цілісних об'єктів означена тільки одна операція — присвоєння. Слід пам'ятати, що присвоєння значення змінній деякого типу запису призведе до присвоєнь значень всім полям цієї змінної. Присвоєння значень змінних типу запису іншим змінним можливе тільки за умови їх однорідності, тобто змінні, що беруть участь у присвоєнні,

повинні мати однаковий склад компонентів та їх типів. Наприклад, для оголошених нижче змінних *11* та 22 присвоєння є коректним.

yar 21.22: recorci

```
            зіпзіппд;
            питЬег: геасії;
          епсі;
Ьеділ
  21:=22:
епсі.
```

Приклад 8.3__

Розглянемо приклад, у якому екземпляри записів обмінюються значеннями. Припустимо, що оголошено тип регзоп і у змінні тапі та тап2 введені значення прибутку. Треба упорядкувати два записи так, щоб у змінній тапі зберігалися відомості про ту людину, доход якої більший.

```
їуре  регзоп=recorcі                    {оголошення типу запису}
        Тіг5І_паше:5І;гіпд;                          {ім'я людини}
        proTiї:reai;                                      {доход}
      епсі:
уаг тап1,тап2,ішр: регзоп:                {екземпляри записів.
                                          що обмінюються значеннями}
Ьеділ
  игіІеІпС'Введіть компоненти записів');
  у/іІН тапі сіо       {ініціалізація компонентів}
  Ьеділ
    ТіГ5І_пате:='Іуапоу';
    геасії п(proTi I);
  ЄП(1;
  milH тап2 сіо
  Ьеділ
    ТігзІ_пате:='Ре1:гоу':
    геасЛп(proTi1;);
  епсі:
  іТ тап1.proTii<тап2.proTiI ІНеп
  Ьеділ                {обмін значеннями між компонентами     }
    Ітр:=тап1;
    тап1:=тап2;
    тап2:Чтр;
  епсі:
  мііЬ тапі сіо          {вивести значення компонентів на екран}
  Ьеділ
    мгіІе"1п(і¹ir5i:_пате);
    мгіІе1п(proTi1:);
  епсі;
  иіі:Ь тап2 сіо
  Ьеділ
    мгі1:е1п(Тігзі;_пате);
    мгіІеІп(proTii):
  епсі;
  геасЛп;
епсі.
```

8.1.3. Масиви записів

У прикладах, що розглядалися раніше, використовувалися лише окремі змінні типу запису. Але в практичних задачах записи частіше використовуються як складові частини масивів, списків, дерев та інших структур. Як приклад такого використання записів розглянемо задачу створення телефонного довідника. **У** цьому довіднику зберігається однотипний набір даних про всіх абонентів: телефонний номер, прізвище й адреса його власника. Отже, телефонний довідник містить набір записів, що мають однакову структуру. Тому найбільш природною формою зберігання інформації телефонного довідника буде масив записів.

Елементи масиву записів обробляються так само, як інші змінні типу запису, зокрема їх можна використовувати в конструкції міііі. При зверненні до полів запису, що є елементом масиву, застосовуються дві операції: індексування ([]) і доступу до компонента (.). Порядок запису символів згаданих операцій є таким:

<1м'я масиву записів>[<1ндекс компонента масиву записів>].<ім'я поля>

Якщо операція індексування [] застосовується до масиву, який є компонентом запису, то маємо синтаксичну конструкцію іншого вигляду:

<ім'я змінної типу запис>.<ім'я поля запису>[<1ндекс компонента масиву>]

Але повернімося до задачі складання й обробки телефонного довідника.

Приклад 8.4

Створимо телефонний довідник, дані до якого користувач вводить із клавіатури. Треба здійснити виведення даних, сортування записів довідника за алфавітом і пошук номера потрібного абонента. Для розробки програми застосуємо метод низхідного проектування, розглядаючи всю програму як набір процедур, що виконують деякі легко означувані операції. Одна з процедур — процедура введення даних, яку назвемо сгеаіе. Ця процедура послідовно заповнюватиме елементи масиву записів. Після введення кожного запису користувачеві буде задане питання **'соп1пие?** у/п' (продовжувати? так/ні). Цикл введення триває доти, доки користувач у відповідь не натисне клавішу N. Виведення даних здійснюватиме процедура **prlnl,** що переглядає та виводить на екран всі записи, поки не буде досягнуто кінця масиву. Дані у довідниках впорядковуються за певним ключем, котрим вважатимемо ім'я абонента.

Сортування записів довідника за іменами абонентів виконуватиме процедура зогЬ, використовуючи при цьому алгоритм сортування методом обміну (бульбашкове сортування), який було розглянуто у прикладі 7.11. В упорядкованому наборі даних можна здійснити швидкий бінарний пошук, алгоритм якого розглянуто у розділі 7.1.2. Процедуру пошуку назвемо **BinSearcli.** Щоб визначити, чи був пошук успішним, скористаємося булевою змінною **Tl ag.** Для зберігання записів довідника оголосимо тип запису ciaTa, що містить у полях рядкового типу **ім**'я абонента, його адресу та номер телефону, а також оголосимо масив **ci i reciory** записів типу ciaia.

```
prodгаш ex8_1:
изез сгі;
lype ciaIa=recorc1                          {тип записів довідника }
       изег_пате:зІппд;                     {прізвище абонента     }
       асісігезБ:зїгіпд;                    {адреса абонента       }
       рИопе:вігі пд;                       {телефон абонента      }
     епсі;
уаг
  ciiгeciory:array [1..10] oT ciaia; {масив записів довідника }
  пишЬегіпІедег;                {кількість записів            }
  тісісііе:іггЬедег:            {індекс середнього компонента}
  пате;зІгіпд;                  {ім'я для пошуку              }
  TIад:Ьooieап;                 (ознака успішності пошуку     }
{========== створення масиву записів  ====================}
 procesiиге сгеаіе;
 уаг кеу;сйаг;                  {КОД натиснутої клавіші       }
     i:inTедег;                 {індекс компонента масиву     }
 Ьеdіп
   \л/гїТе!п( 'епТег ciirecTory ciaia');
   і :-0:
   гереаі
      і:=і+1;                   {перейти до наступного запису}
      мі їй c1irecIory[i] сіо
      Ьеdіп                     {ввести дані                  }
         мгіїеСпате: ');    геасПп(изег_паше):
         мгіІе('асісігезз:  '); геасіїп(асісігезз);
         мгіїеСІеІ;  ');     геасііп(рйопе);
      епсі;
      мгіІе"Іп('сопїіпие?  у/п');
      кеу;=геас1кеу;
   ипііі  (кеу='п')ог(кеу='N'};
   питЬег;=і;                  {запам'ятати кількість записів}
 епсі;
{============== виведення масиву  ========================}
 procesiиге ргіпі;
 уаг  і: іпіедег;
 Ьеdіп                  _________     ______      _____
   мгіїеІпС-- изег паше        асісігезз      Іеіерйопе     ');
   Тог  і :=1  Io  пишЬег сіо
   Ьеdіп
     мііїй c1irecIory[i] ОО
     Ьеdіп
       мгіІеС     ' ,изег_пате);
       мгіІеГ     '.асісігезз);
       мгіїеС     ' ,рйопе);
     епсі;
     МГІІеІп;
   епсі;
  епсі;
```

•риз

:(,згшідиоз 0} А^ /Ов рив цзшц. иоі 353 ззалсі,)и[3}им
:(.рипсу. аои риооал,)и|.а:іим аз|а
(аиоцсі",<-аиоцсі
, 'аиіви^азп' ,<-аіиви иазп , "а[ррш', і.,)и[з^.им
иацр. бвц. л
ор [з [рр|.ш]/Сиоазз-і Ф
: (издшпи' і)ірива$иі.д
:(ашви)и|.рвз.л
:(,:цзивзз ло^ зшви ла}иа,)аіим
{353 /Сіліавітх **ОІАНЗШВН** ан ихой 'я/тои ихваисіоіаои} }вас!аи

:(диоз иагцв зриоззи ^о /Свилв,)иіаіим

:}иисІ
:(,зриоози апсіиі. ^о /Сеиие,)и[а^.
:з^ваиз
: (,/Осцзаиі.р аиоцсіз[9^.)и[3}им
иі.6ад
{========== енесІ.іосІи днаоцш==========;
:риз
:риа
: (}цби'х+а|.рр!-Ш)ірива$и|.д аз[а
(1-а[ррш'аіа[)цзива5и!.а
иащ ашви иазп'[а[ррш]]/Сиоіоа.іір>ашви л а$|а
апл}=:6ви
иаі)} зшви-^азп'[з[ррш]А'иоіззиір=зшви л
••І л(.р (Щби+:ца|>:а[рр!.ш
иібая
аз [а
аз|_в^=:6в[^
иащ щ6исца[л
и|.6ад
: (^бапиі.:ацби':уз[)цзавз$и(.д алпрааоисі
{==========!_ аизви А Лзиивєм Лтои и к неї в н і. 9==========}
:риз
:риа
:риз
:сІиц=: [Г]/Оо:ра.ир
:[р]£іоаоз.ир=: ІЧ]/Оо:ра.пр
:|Ч]/Оо:раи!.р=:скц
и(.6ад
иаца зшви^а$п'[Г]/иорзлір<ашеи-из5П'[і]Лиоіззиі.р л
иі.6ая
ор иадшпи о} і+и:Г .Ісу
ор і-издшпи 0} х-Ч л<У
и і-бад
: иабаяи і.: Г' і.
:вавр:сІші ^л
::роз аипразоисі
{========== ноілавфі/в **ВЕ** аіоиивє Лаиовн вннва/Сісіоз=========}

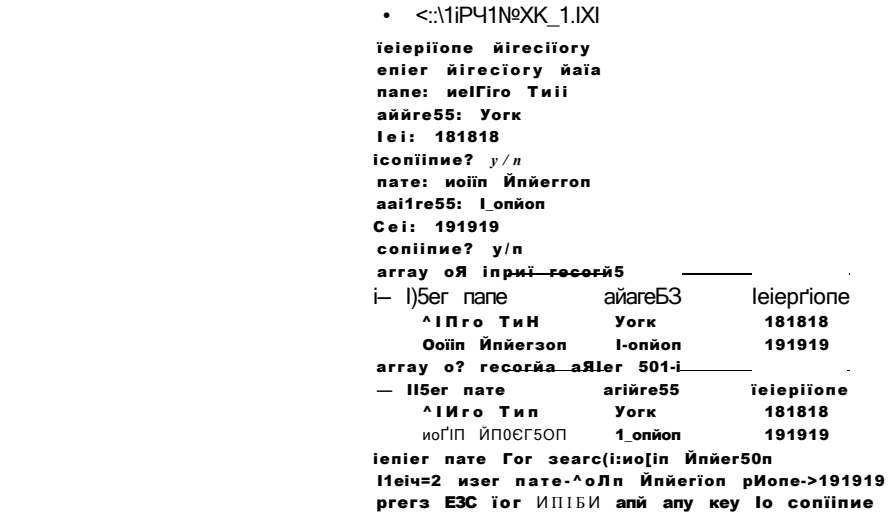


РИС. 8.2. Результати роботи програми ex8_1. Телефонний довідник

8.1.4. Записи з варіантами

Дані структурованого типу запису, який було розглянуто у попередньому розділі, складаються з фіксованої кількості компонентів. Фіксованість структури записів обмежує область їх застосування: такі записи можуть зберігати лише однотипні набори даних. Тому в мові Разсаі є різновид записів, що об'єднують довільну кількість компонентів. Припустимо, треба створити масив записів з інформацією про викладачів і студентів навчальних закладів. Для викладачів суттєвими характеристиками є науковий ступінь, кількість наукових праць, дисципліни, що вони їх викладають, тощо. Для студента суттєвими ознаками є середній бал, форма навчання (денна або заочна) і т. ін. Водночас викладач і студент мають спільні характеристики, такі як прізвище та плата за роботу (заробітна плата або стипендія). Отже, запис з інформацією про учасника навчального процесу матиме *фіксовану* та *варіантну частини.* Такий запис називається *записом із варіантами.* Доступ до полів варіантної частини здійснюється за певних умов. Запис містить спеціальне *поле ознаки,* і його значення вказує, які поля варіантної частини є активними. Оголошення запису з варіантами має такий синтаксис:

```
Iуре <ім'я типу> = гесогсі
                     <фіксована частина>;
                     <варіантна частина>;
                   ЄПС;
```

Фіксовану частину оголошують у звичайний спосіб, як перелік полів та їх типів. Варіантна частина формується за допомогою оператора сазе. Якщо він зна-

ходиться усередині запису, то називається оператором варіанта. У ньому зазначається ім'я та тип поля ознаки, а також міститься перелік наборів полів. Кожен набір полів супроводжується деяким значенням, константою вибору. Якщо значення поля ознаки дорівнює деякій константі вибору, активізується певний набір полів. Наведемо синтаксис оголошення варіантної частини запису:

```
case <ім'я поля ознаки>:<тип> of
  «константа вибору!>:(<список полів>);
  <константа вибору2>:(<список полів>);

end;
```

Список полів - це перелік розділених символами крапки з комою записів вигляду <ім'я поля>:<тип>.

Приклад 8.5_

Оголосимо запис person, що містить відомості про викладача або студента, та зобразимо схематично його структуру (рис. 8.3). У цьому записі поля name та finance становитимуть фіксовану частину, а поля - sedgee, pitber, tagk, Togt - варіантну. Поле Iad буде полем ознаки.

```
type person=record
       PIO:5i;gipd:            {прізвище викладача або студента}
       Tinance:reai:           {заробітна плата або стипендія  }
       case Tad: (TeacIeg. zTiclenl;) of
                               {дані про викладача:            }
         ieacIeg:(siedgee:zi:gipd;   {науковий ступінь викладача}
                  pitber:iniedeg); {кількість наукових праць   }
                               {дані про студента:             }
         Zius1en1::(shagk:rea1:      {середній бал успішності    }
                  Togsh:zigipd);     {форма навчання студента    }
     end;
```

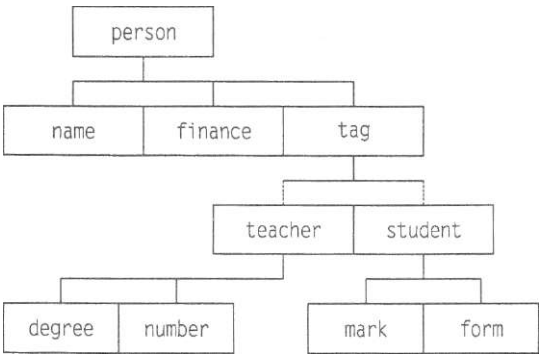


Рис. 8.3. Структура запису з варіантами

Оголошення записів із варіантами має відповідати певним синтаксичним правилам:

- запис може мати тільки один оператор варіанта ca$e;
- варіантні компоненти вказують після фіксованої частини запису;
- варіантна частина може мати довільну кількість варіантів (альтернатив);
- слово encl завершує оголошення всього типу, а не лише його варіантної частини;
- компоненти кожного варіанта записуються у круглих дужках;
- усі ідентифікатори полів усередині одного запису мають бути унікальними;
- тип поля ознаки для вибору варіанта з варіантної частини має бути перелічуваним;
- константа вибору може дорівнювати лише значенням, які мають тип поля ознаки.

Наступний приклад демонструє використання записів із варіантами.

Приклад 8.6

Розглянемо задачу створення журналу обліку викладачів і студентів. Про викладача є такі відомості: прізвище, науковий ступінь, посадовий оклад і кількість наукових праць, а про студента — такі: прізвище, середній бал, форма навчання (денна або заочна) і розмір стипендії. Необхідно ініціалізувати список викладачів і студентів та вивести дані про тих людей, дохід яких (посадовий оклад або стипендія) перевищує заданий рівень.

Відомості про викладачів і студентів подамо у вигляді масиву записів. Оскільки викладачі та студенти мають різні реквізити, то слід використовувати записи з варіантами. Фіксована частина запису має містити спільні для студента та в'икладача характеристики, тобто прізвище та доход. Інші характеристики відображатимуться полями варіантної частини запису. У процедурах введення та виведення даних використаємо значення поля ознаки для активізації потрібного варіанта запису. Введення значення 1 активізує варіантну частину запису викладача, значення 2 — запису студента.

Сформульовану вище задачу розв'язано у програмі ex8_2. Пропонуємо читачеві доповнити цю програму процедурами визначення викладачів із найбільшою кількістю наукових праць і пошуку найкращих студентів.

```
ргодгаш ex8_2;
Иб€5 сгї;
lype регзоп=recorci
       паше:5Їгіпд:            {прізвище викладача або студента}
       геуепие:іпїедег;        {заробітна плата або стипендія  }
       сазе Іад: (їеасИег. зіисіепі) оТ
         IeacHer: (сіедгее:зїгіпд:питЬег:іпїедег);
         зіисіепі: (шагк: іпїедег: Тогш:зІп пд);
     encl;
уаг
1ізі:агтау [1..10] оГ регзоп;    {список викладачів і студентів}
  і:іпіедег;                     {індекс компонентів масиву     }
  ІєУєІ:іпіедег;                 {рівень доходу                 }
```

```
|========== введення масиву викладачів 1 студентів
ргосесіиге ІприТ;
уаг 5ідп:іпіедег;              (ознака категорії ЛЮДИНИ          }
    сМ: сіїаг:                 (ознака завершення циклу
Ьеділ
  сігзсг;
  і :=0;                   (лічильник елементів масиву записів}
  гереаі
    і :=і+1;                   (ввести наступний запис          }
    мїїі ііві[і] сю
     Ьеділ
       мгіїеСіприї Type оТ регзоп: 1-їеасНег, г-зіисіепі ');
       геасПп(зідп);               (ввести ознаку об'єкта     }
       іТ зідп=1 ІЬеп їад:=ІеасНег еізе їад: =5ИисІеп1;;
       сазе іад оТ
         ІеасИег:Ьеділ             (ввести дані про викладача}
           мгііеСпаше '): геасПп(пате):
           мгіі;е([1] зсіепііТіс сіедгее '); геасПп(сіедгее):
           игіїеС'питЬег оТ зсіепТіТіс могкз [1]);
           геасіїп(питЬег);
           мгіі:е([1] геуепие '); геасіїп(геуепие);
         епсі;
         зИисІепТ: Ьеділ             (ввести дані про студента}
           мгііеСпате '); геасііп(пате);
           мгіІе('геуепие '); геасііп(геуепие);
           игіІе('ауегаде тагк '); геасіїп(тагк):
           мгіїеСТогт оТ есіисаііоп '); геасіїп(Тоггп):
         епсі;
       епсі;                                          (епсі оТ сазе}
    епсі;                                             (епсі оТ міііі}
    мгіІе1п('соп іипие? у/п[1]);
    сіі;=геас1кеу;
  ипїіі сИ='п':       (натиснути 'п' для завершення введення)
  мгіІеС'епїег геуепие ІЄУЄІ ');
  геас11п(1еуе1);                    (введення рівня доходу}
епсі;                                        (епсі оТ ргосесіиге}
{=== ==ВИВЄСТИ масив викладачів і студентів
ргосесіиге Оиїриї;
уаг ^ Ипїедег;
Ьеділ
  Тог з :=1 Іо і сіо
    мШі ііз1СЛ сіо
      іТ геуепие>1еуе! їНеп
        сазе їад оТ
          ТеасИег; Ьеділ
                     мгіїе1п( 'ТеасІїег    .пате.'  '.сіедгее,'  '.питЬег.'  '.геуепие);
                     геасії п;
                   епсі;
          зіисіепі: Ьеділ
                     мгіїеІпС 'зіисіепі:    .пате.     .геуепие,'  '.тагк,'  '.Тогт);
                     геасії п;
                   епсі;
        епсі;                                  (епсі оТ сазе      }
епсі;                                          (епсі оТ ргосесіиге}
```

```
{==================== головна програма ====================}
Ьеділ
  сігзсг;
  Іприі;
  Оиіриї;
  геасії п;
епсі.
```

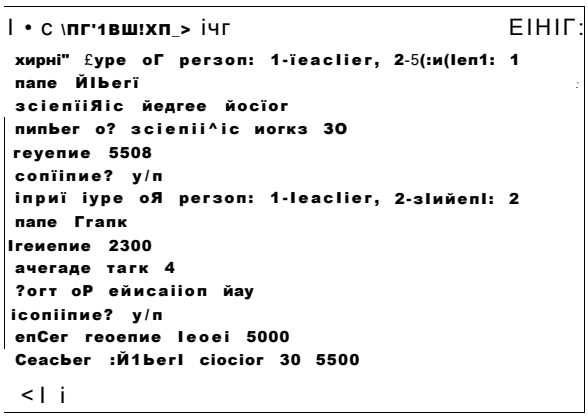


РИС. 8.4. Результати роботи програми ex8_2. Журнал обліку

8.2. Множини

Математичне поняття множини широко використовується в задачах, для яких існує ефективне програмне розв'язання. Так, у багатьох комбінаторних задачах серед усіх підмножин деякої множини необхідно знайти ті, які задовольняють певну умову. При розв'язанні задач на графах користуються поняттями множин вершин і ребер, а сам граф відображається на площині як множина точок і ліній. У мові Разсаі означений відповідний тип даних, який буде розглянуто в цьому розділі.

8.2.1. Поняття множини та множинного типу даних

Множиною в мові Разсаі називається скінченний набір однотипних даних. Об'єкти, з яких складається множина, називаються її *елементами.* Число, що дорівнює кількості елементів множини, називається її *потужністю.* Від масиву множина відрізняється відсутністю індексації її елементів, а від запису - тим, що елементи множини є однотипними і не позначаються ідентифікаторами.

Найпростішим прикладом множини є неіменована множина-константа, що записується у квадратних дужках як перелік однотипних елементів. Наприклад, [0.1] - це множина, що складається з двох цілочислових елементів, а['a'..'2'] — множина, що містить усі маленькі літери латинського алфавіту - 26 символьних

елементів. Максимально допустима потужність множини в мові Разсаі становить 256, а мінімально допустима — 0. Множина нульової потужності, тобто множина, що не містить жодного елемента, називається *порожньою* і в математиці позначається символом 0. У мові Разсаі порожня множина позначається як порожній перелік, тобто символами [].

Зауважимо, що порядок елементів у переліку є несуттєвим і кожен елемент входить до складу множини один раз. Тому, скажімо, множина `[1,2,3,1,2.3]` еквівалентна множині `[1,2,3]`, а також множинам `[1,3,2]`, `[2,3,1]`, `[2,1,3]`, `[3,1,2]` і `[3,2,1]`. Крім констант-множин у Разсаі-програмах можна використовувати й змінні множинного типу даних, до розгляду якого ми і перейдемо.

Множинний тип — це структурований тип даних, допустимими значеннями якого є деякі множини. Отже, сукупність допустимих значень такого типу даних — це є множина множин, яка не записується в оголошенні типу безпосередньо, але може бути визначена з його оголошення за певним правилом. Це правило легко зрозуміти, коли розглянути синтаксис оголошення множинного типу даних:

```
їуре <ім'я тилу> = зеї оТ <базовий тип>;
```

Як бачимо, оголошення множинного типу даних вимагає визначення для нього деякого *базового типу.* Базовим може бути будь-який перелічуваний тип, крім Іпіедег та [_опдІпі, а значенням множинного типу може стати довільний набір значень базового типу. Таким чином, допустимими значеннями множинного типу даних є всі можливі підмножини множини значень певного базового типу. Наведемо приклади. Припустимо, є таке оголошення множинного типу TypeA:

```
їуре TypeA - зеі: оТ 2. .4;
```

Тоді множина допустимих значень типу TypeA міститиме такі набори цілих чисел: {2,3,4}, {2,3}, {2,4}, {3,4}, {2}, {3}, {4}, 0. Отже, із значень трьохелементного базового типу можна утворити вісім множин. Постає запитання: скільки різних множин можна утворити на основі гс-елементного базового типу? З відомих математичних фактів випливає, що таких множин може бути утворено *2".*

Щойно згадана величина п, яка дорівнює кількості елементів базового типу, не може перевищувати 256, оскільки потужність будь-якої множини в мові Разсаі не може бути більшою за це число. Саме тому неприпустимо використовувати такі базові типи, як ІпТедег та I_опдІп1:.

8.2.2. Оголошення змінних множинного типу

Оголошення змінної типу масиву або типу запису може бути відокремленим від оголошення відповідного типу або поєднаним із ним. Таку саму ситуацію маємо і зі змінною множинного типу даних: існує два різних синтаксиси її оголошення. А саме:

```
уаг <ім'я змінно1>:5€I оТ сбазовий тип>;
```

та

```
уаг сім'я змінної>:сім'я множинного типу>;
```

При оголошенні множинної змінної першим способом неявно оголошується неіменований множинний тип даних, а другий спосіб потребує попереднього оголошення множинного типу в розділі Iype. Наведемо приклади оголошення констант і змінних множинного типу, означуючи самі типи явно, в розділі їуре, або неявно, в розділі уаг:

Приклад 8.7

```
їуре
  йідіїз = зЄЇ оГ 0..9;                    {тип множини цифр    }
  [_Є«ЄГ5 = 5ЄЇ оТ 'А' . . 'Г ;             {тип множини латиниці}
  йау = (5ип, Моп, Тие, Месі. ТПїи, Ргі, 5аІ); {тип множини днів }
  УогкОау = Моп..Ргі ;                          {тип діапазону   }
сопзі
  ЕуепОідПз: Оідіїз - [0, 2, 4, 6, 8];
  Уомеїз: ЬеНегз - ['А', 1Е1. 'Г, '0', 'ІГ , 'У'];
  НехОідііз: зеї оТ '0'..'2' = ['0'.. '9', 'А'..'Р'. 'а'.. 'Г]:
уаг
  зутЬої :зеІ оТ сіїаг;        {множина символів                }
  меек:зеі: оТ Бау;            {множина днів від 5ип до 5аІ     }
  МогкМеек: зеї: оТ ИогкОау:   {множина днів від Моп до Ргі     }
  пишЬег:зеі; оТ Ьуіе;         {множина цілих однобайтових чисел}
  ініzеі:зе1: оГ 0. .100;      {множина чисел від 0 до 100      }
```

8.2.3. Операції над множинами

У мові Разсаі визначено аналоги майже всіх основних теоретико-множинних операцій та відношень. Базовим відношенням у теорії множин вважається відношення належності (елемент *x* належить множині *A),* що символічно позначається як *x* є *A* (рис. 8.5, 3). Перевірка істинності цього відношення у Разсаі-програмі здійснюється оператором і п. Використовуючи відношення належності, можна означити основні бінарні операції над множинами, якими вважаються операції об'єднання, перетину та різниці. Нагадаємо їх означення.

В результаті *об'єднання множин* утворюється множина, що складається з елементів, які належать хоча б одній з даних множин (рис. 8.5, *а):*

$$A[]B = \{x\backslash x \quad \text{Є Л У І} \quad eB\}.$$

Результатом операції *перетину множин є* множина, що складається з елементів, які належать кожній з даних множин (рис. 8.5, *б):*

$$AГ\backslash B=\{x\backslash x \quad eA\&x \quad \epsilon B).$$

Різниця множин A і *B* (доповнення *A* у *B)* - це операція, в результаті виконання якої утворюється множина, що складається з тих елементів множини *A,* які не належать множині *B* (рис. 8.5, б):

$$A\backslash B \quad =\{x\backslash x \quad \text{є} \quad A\&x \quad iB\}.$$

Крім відношення належності елемента до множини в математиці використовують ще відношення множин, аналогами яких у мові Разсаі є визначені для множинного типу даних операції порівняння. Основні такі відношення визначають, чи збігаються множини та чи є одна з них підмножиною іншої. Нагадаємо, що множина ***A*** є підмножиною множини ***B*** (іншими словами, ***A*** міститься у ***B,*** або ***B*** містить ***A),*** якщо кожний елемент ***A*** є елементом ***B*** (рис. 8.5, ***г):Ас B {x єA=> => x є B}.*** Дві множини є рівними, якщо кожна з них є підмножиною іншої: ***A-BoAcB&Bc A.*** Зрозуміло, що результат операцій порівняння множин у мові Разсаі, як і результат застосування оператора іп, має логічний тип.

а б в г д

Рис. 8.5. Операції над множинами та відношення множин: об'єднання (а); перетин (б); різниця (в); входження (г); належність (д)

Як уже згадувалося, всі означені в мові Разсаі операції над даними множинного типу мають відповідники у математиці. У табл. 8.1 перелічено всі такі операції та Наведено їх запис як у математичній, так і у Разсаі-символіці.

Таблиця 8.1. Операції над множинами

Математичний запис	Запис на мові Pascal	Семантика
A = B	A = B	Множини ***A*** і ***B*** збігаються
A*B	A <> B	Множини ***A*** і ***B*** не збігаються
A^B	A >= B	Множина ***B*** є підмножиною множини ***A***
AcB	A <= B	Множина ***A*** є підмножиною множини ***B***
x є A	x іп A	Значення л: належить множині ***A***
A и B	A + B	Об'єднання множин ***A*** і ***B***
AnB	A * B	Перетин множин ***A*** і ***B***
A\B	A - B	Різниця множин ***A*** і ***B***

Операнди наведених у табл. 8.1 операцій мають задовольняти певні обмеження. А саме, операнд x оператора x і п A має належати базовому типу множини A, а операнди решти операцій мають бути множинами з однаковим базовим типом.

Важливі окремі випадки операцій об'єднання та різниці множин реалізовано у вигляді бібліотечних процедур Incl иcie та Excиcie, що здійснюють включення елемента до множини та вилучення його із множини відповідно. Наведемо прототипи цих процедур.

```
IncиcIe(yar 5;i):
Exclиcie(yar $;i);
```

Тут 5 — це змінна множинного типу; і — елемент, який включають до множини 5 або видаляють з неї. Тип і повинен бути базовим типом множини 5.

Завершуючи розгляд операцій над множинами, зазначимо, що, як і змінним усіх інших типів мови Разсаі, змінній деякого множинного типу можна присвоювати значення виразів того самого типу. Слід зауважити, що одноелементна множина може бути утворена і шляхом запису імені цієї змінної базового типу у квадратних дужках. Тому коректним є таке присвоєння:

<ім'я змінної типу множини>:=[<ім'я змінної базового типу>];

Подальші модифікації значень змінної базового типу, використаної у такому присвоєнні, не призведуть до модифікації значень відповідної множинної змінної. Наприклад, після присвоєнь x:=1; A:=[x]: x:=2; значення єдиного елемента множини А дорівнюватиме одиниці.

Приклад 8.8 демонструє застосування операцій над множинами. У програмі формуються дві множини: множина символів епзешЬІе та множина чисел ОідН. Множина епзешЬІе спочатку є порожньою, введені користувачем символи додаються до неї доти, доки не буде введено символу ' п'. При цьому введені знаки питання замінюються знаками оклику. З множини Оідіі, яка спочатку містить цілі числа від 1 до 10, введені користувачем елементи вилучаються доти, доки не буде вилучено одиниці.

Приклад 8.8_

```
маг епзешЬІе :зеі: оТ сііаг:
    •ідії:5е1; оТ Ьуїе:
    сіксіїаг:
    питЬег:Ьуіе;
Ьеділ
  епзетЬ1е:=[];
  гереаї
    мгіїе1п('Епіег зутЬої. Епіег "п" Іо Тіпізії');
    геасПп(сЬ):
    еп5етЬ1е:=епзетЬ1е+[сІі];
    іТ сЬ='?' їИеп епзешЬІе:=епзетЬ1е+['!'];
  ипїіі 'п' іп епзешЬІе:
  Оіді-Ь:=Е1. .10]:
  гереаі;
    мгі1:е1п('Епїег питЬег. Епїег 1 іо Тілібь[1]);
    геасії п( питЬег);
    0іді1::=0ідіІ-[питЬег]:
  ипїіі поШ іп Оідії);
епсі.
```

УВАГА

Змінні множинного типу не можна використовувати як аргументи процедур введення геасі або геайіп і процедур виведення \мгі(е або УУПІЄІП. ВВОДИТИ **та виводити множини можна лише поелементно.**

У прикладі 8.9 реалізовано операції введення та виведення множини. Значення, що були введені до змінної сії базового типу, додаються до множини зутЬої

операцією об'єднання (+). Під час виведення множини зутЬої переглядаються всі елементи її базового типу, які можуть бути введені користувачем. Кожен із них перевіряється на належність множині.

Приклад 8.9_

```
ргодгат ех8_3;
уаг
   зутЬої:зеІ оТ сНаг;
   сИ:сііаг:
Ьеді п
  мгіїеІпС'ргі пі еіетепїз Тгот зеї');
  зушЬо!:=[];
  гереаі;
    «гііеІпС'епїег зушЬої. ргезз "п" То Тіпізії'):
    геасііп(сіі):
    зутЬої:=зутЬо!+[сЬ];
  ипііі сЬ='п';
  Тог сИ: = '0' То 'я' сіо
    іТ сЬ іп зутЬої ТЬеп мгіТеССЬ,' ');
  геасії п;
епсі.
```

```
[• (Іпас»і«С:\ВР\ПИи:ХП 1.ГХГ)
prini еіетепїг; Гтот зеї
enier зутЬої, РГЄ55 "П" ю £ІПІЗЬ
Ч
enier зутЬої, ргезз "П" 10 Гіпізь 1
и
enCer гутЬоІ, ргЄ55 "П" іо Нпізь |
р
enïer зутЬої, рГЄ53 "11" Ьо ГІПІЗІ1 1
y
enier зутЬої, рГЄ55 "п" іо ГІПІЗЬ !
п
е п ^ і- ш

« п
```

РИС. 8.6. Результати роботи програми ех8_3. Виведення множини

8.2.4. Зображення множин в оперативній пам'яті

Значеннями однієї змінної множинного типу можуть бути множини різної потужності, але потужність будь-якої з цих множин не може перевищувати кількості елементів базового типу. Тому обсяг пам'яті, виділеної для зберігання множинної змінної, визначається базовим типом цієї змінної. Кожне значення базового типу зображується одним бітом. Порядок розташування бітів відповідає визначеному у базовому типі порядку (нагадаємо, що базовим типом множини може бути лише порядковий тип даних). Належність елемента до множини позначається одиничним значенням відповідного біту, а відсутність елемента у множині — нульовим значенням. Розглянемо приклад.

Приклад 8.10

Нехай у програмі оголошено деякий множинний тип, базовим типом якого є набір 26 символів 'A'..'1'. Під час компіляції програми змінній такого множинного типу буде виділено 26 бітів пам'яті. На початковому етапі виконання програми ці біти будуть заповнені нулями, а під час присвоєння змінній деякої множини великих латинських літер нулі у відповідних бітах буде замінено одиницями. Наведемо код демонстраційної програми та побітове зображення утвореної в цій програмі множини т.

```
Type I_еиег$ = зеї оТ 'A'.. 7';
уаг ш: I_е«егз;
Ьеділ
   ш:=['Р', 'А'. '5', 'С', 'А'. Ч.']:
   Тог сіі: = 'А' їо 7' сіо
     іТ сії іп ш їИеп мгіїе(сИ,' ');
епсі.
```

1	0	1	0	0	0	0	0	0	0	0	і	0	0	0	1	0	0	1	0	0	0	0	0	0	0

Зіставлення бітів і значень елементів базового типу

А	В	с	Б	Е	Е	С	н	І	}	к	ь	м	N	0	р		к	5	Т	II	V		Х	У	*г*

У результаті виведення значень елементів множини отримаємо набір символів 'А С І. Р 5'. Цей приклад ілюструє взаємозв'язок особливостей зображення множин в оперативній пам'яті з такими їх властивостями, як відсутність повторень і невпорядкованість елементів.

Зображення множини у вигляді бітового масиву зводить операції об'єднання, перетину та різниці двох множин до порозрядних логічних операцій над послідовністю бітів. Наприклад, перетин множин виконується шляхом логічного множення бітів:

```
уаг т,п:$еі: оТ 1. .9;
```

т:= [2, 4. 5, 6. 7, 8. 9];	0 1 0 1 1 1 1 1 1
п := [1. 3, 5. 7, 9];	1 0 1 0 1 0 1 0 1
т*п = [5. 7. 9]	0 0 0 0 1 0 1 0 1
Значення елементів множин	1 2 3 4 5 6 7 8 9

Висновки

- Запис - це структурований тип даних, що є об'єднанням фіксованої кількості змінних одного або декількох типів. Змінні, що входять до складу запису, називаються його полями. Головна відмінність запису від масиву полягає у можливості інтегрування в один запис різнотипних компонентів.

- Звернення до полів запису здійснюється через складене ім'я, що містить розділені крапкою імена змінної типу запису та компонента запису.
- Тип запису може бути типом елементів масиву. У разі звернення до полів запису, що є елементом масиву, спочатку застосовується операція індексування ([]), а потім — операція доступу до компонента (.).

4- Записи з варіантами складаються з необов'язкової фіксованої та обов'язкової варіантної частин. Поля фіксованої частини є доступними завжди, а доступ до полів варіантної частини здійснюється за певних умов. А саме, запис містить спеціальне поле ознаки, значення якого показує, які поля варіантної частини є активними.

- Множиною в мові Разсаі називається скінченний набір однотипних даних. Об'єкти, з яких складається множина, називаються її елементами, а кількість елементів множини - її потужністю. Від масиву множина відрізняється відсутністю індексації її елементів, а від запису - тим, що елементи множини є однотипними і не позначаються ідентифікаторами.

+ Множинний тип — це структурований тип даних, допустимими значеннями якого є деякі множини. Сукупність допустимих значень такого типу даних - це множина множин.

- Синтаксис оголошення множинного типу даних вимагає визначення для нього певного базового типу. Базовим може бути будь-який перелічуваний тип, крім ІгтЬедег і Іопдіпі, а значенням множинного типу може стати довільний набір значень базового типу. Таким чином, допустимими значеннями множинного типу даних є всі можливі підмножини множини значень певного базового типу.
- У мові Разсаі означено аналоги таких теоретико-множинних операцій, як об'єднання, перетин і різниця, а також операцій порівняння множин. За допомогою оператора і п можна визначити, чи належить елемент множині.

Контрольні запитання та завдання

1. Дайте означення запису як типу даних.
2. У чому полягає відмінність запису від масиву?
3. Як здійснюється доступ до компонентів запису?
4. Які операції можна виконувати над даними типу запису?
5. Як здійснюється доступ до компонентів запису, що є елементом масиву?
6. Як визначається обсяг пам'яті, необхідний для зберігання запису?
7. З якою метою використовують записи з варіантами?
8. Дайте означення типу даних «запис з варіантами».
9. Які переваги дає використання оператора приєднання міі:Н?
10. Що являє собою об'єкт «множина» в мові Разсаі?

11. Як визначається множина допустимих значень множинного типу даних?
12. Які типи даних можуть бути базовими для множинного типу?
13. Які операції можна виконувати над множинами?
14. Як множини зображуються в оперативній пам'яті?

Вправи

1. За наведеним нижче фрагментом програми визначити істинні твердження.

```
yar a: Ьуіе;
    гар: recorci
        a: Ьуіе;
        Ь: геаі;
    епсі;
```

   1.1. Присвоєння значення змінній а приведе до автоматичного присвоєння полю а запису гар того самого значення.

   1.2. Компілятор розглядає змінну а та поле а запису гар як різні змінні.

   1.3. Компілятор видає помилку.

2. Визначити коректні оголошення множини.

```
M:5ei oT іпїедег;
M;5e1: oT cHar:
M:зеі: оГ 1. .256;
M:5ЄЇ oT '0 '.. '9' ;
M;зеі: oT 0. .9;
M:зеі: oT (гесі.дгееп);
M: 5еї oT геаі;
```

3. Які з наведених нижче множин еквівалентні множині [2,3.4,5]?

```
[2,3,4,4,5];
[3,2,4,5];
[2..5];
[5..2].
```

4. Зроблено такі оголошення:

```
yar A,B: зеі; oT cHar; x: cЬar:
```

   У яких виразах допущено синтаксичних помилок?

```
B:= A + x;
[B]:- [A] + [x];
B:= A + [x];
B:= [A] + x;
```

Задачі

1. Оголосити тип запису, що містить відомості про прізвища та адреси людей. Ввести два масиви таких записів. Записи, які є в першому масиві та яких немає у другому масиві, скопіювати до третього масиву.
2. Задано два масиви записів. Записи першого масиву містять відомості про прізвища людей та їх адреси, другого — про прізвища та номери телефонів. Скласти програму, що за цими двома масивами формує третій масив, записи якого мають такі поля: прізвище, адреса, номер телефону.
3. Заданий масив записів, що містять такі відомості, як прізвища студентів і розмір їх стипендії. Необхідно вилучити з цього масиву записи всіх тих студентів, що отримують стипендію, нижчу за середню.
4. Створити масив записів із варіантами для зображення бібліотечного каталогу, в якому є дані про книги, журнали та газети. А саме, про книгу відомі її назва, прізвище автора та рік видання, про журнал — його назва, номер, рік видання та перелік статей із прізвищами авторів. Газети ідентифікуються так: назва газети, її номер, дата виходу, перелік статей із прізвищами авторів. У створеному масиві здійснити пошук робіт автора, прізвище якого введено з клавіатури. З масиву видалити всі газети, видані до 1995 року.
5. Задано рядок символів. Визначити кількість різних символів, що не є літерами або цифрами, і вивести їх на друк, використавши множини.
6. Сформувати множину всіх цілих чисел з діапазону від 1 до 256, які є сумами квадратів двох невід'ємних цілих чисел. Вивести цю множину.
7. Сформувати множину всіх простих чисел, що належить діапазону від 1 до 256. Вивести цю множину.
8. Визначити кількість різних цифр, що містяться в десятковому записі кожного елемента масиву натуральних чисел. Використати множини цифр.
9. Отримати всі перестановки заданої я-елементної множини. (Перестановка множини - це певним чином впорядкований набір її елементів.)
10. Задана множина символів. Скласти програму, що визначає всі її підмножини, які містять парну кількість елементів.
11. Множина *Аз2п* елементів поділена на підмножини *В* і *С,* кожна з яких містить по *п* елементів. Визначити всі перестановки елементів з множини *А,* в яких елементи з *В* і *С* чергуються.
12. Навчальний план підготовки фахівців містить відомості про назви дисциплін та кількість навчальних годин з кожної. Вибрати з навчального плану будь-який блок дисциплін, що складається не менш ніж з 5 дисциплін, що мають сумарний обсяг 756 годин або більше (якщо такий блок є).
13. Довідник продуктів містить назву, калорійність і ціну одного кілограма кожного продукту. Скласти всі можливі меню, сумарна калорійність кожного з яких буде не менше, а загальна вартість — не більше від введених користувачем величин. Меню — це перелік продуктів із зазначенням їх кількості.

Розділ 9
Файлові структури даних

- Поняття фізичного і логічного файла
- Різновиди логічних файлів у мові Разсаі
- Технологія обробки файлів
- Механізм буферизації даних

9.1. Фізичний і логічний файли

Дані, що використовувались у задачах із попередніх розділів, існували протягом одного сеансу роботи певної програми. Такі дані зберігаються в оперативній пам'яті комп'ютера. Проте більшість програм оперує із даними, що залишаються доступними як після завершення роботи програми, так і після перевантаження комп'ютера. Такі дані зберігаються на дискових накопичувачах у вигляді *файлів.*

Поняття файла можна розглядати з двох точок зору. З одного боку, файл — це іменована область на зовнішньому носії інформації, що містить довільні дані. Файл у такому розумінні називають *фізичним* файлом, тобто таким, що існує фізично на матеріальному носії інформації. З іншого боку, файл — це одна із структур даних, що використовується у програмуванні. У такому розумінні файл називають *логічніш,* тобто таким, що існує в певній програмі як абстракція.

Файл як фізичний об'єкт є послідовністю байтів. Фізичний файл характеризується іменем, що його ідентифікує. Розмір файла може бути довільним і обмежується лише ємністю пристроїв зовнішньої пам'яті.

Файл як логічний об'єкт є послідовністю значень певного типу, тобто він складається з однотипних компонентів. Отже, файл — це структурований елемент даних. І тому цілком природним є те, що в мові Разсаі для зображення файлів визначені стандартні структуровані типи даних і можуть бути оголошені змінні цих типів. Оскільки компоненти файла належать до одного типу, то структура логічного файла нагадує структуру масиву. Але можна назвати й суттєві розбіжності між цими структурами даних. А саме:

- під час оголошення масиву слід визначити кількість його елементів. Під час оголошення файлової змінної розмір файла невідомий;

+ розмір масиву, на відміну від розміру файла, не може змінюватися під час роботи з ним;

- для доступу до елементів масиву застосовують індексацію, а для доступу до компонентів файла - покажчики на поточний компонент;

4 нумерація елементів масиву виконується від певної нижньої до певної верхньої межі індексу. Компоненти файла нумеруються починаючи з нуля. Наприкінці фізичного файла записується керуючий символ #26 (OrI+2!), що використовується як ознака завершення відповідного логічного файла;
- компоненти файла можуть належати до будь-якого типу даних, окрім файлового. Тип елементів масиву може бути і файловим.

9.2. Технологія роботи з **файлами**

Робота з файлом у мові Разсаі складається з таких етапів: оголошення файлової змінної; зв'язування файлової змінної з іменем наявного файла або файла, що створюється; відкриття файла; обробка файла; закриття файла. Ці етапи розглядатимуться в розділах 9.2.1-9.2.5.

9.2.1. Типи файлів і оголошення файлових змінних

Файли класифікують за типом компонентів і за методом доступу до них. За типом компонентів розрізняють *текстові* та *бінарні* (двійкові) файли, а за методом доступу — файли *послідовного* і *прямого доступу.* Текстові файли призначені для збереження текстів (наприклад, текстів Разсаі-програм), а бінарні файли використовуються для збереження даних різних типів. Відмінності між послідовним і прямим доступом до компонентів файлів буде роз'яснено у розділах 9.2.4 і 9.2.5.

Текстовий файл є сукупністю символьних рядків змінної довжини. Кожен рядок завершується *маркером кінця рядка* — спеціальною парою керуючих символів: #13 (повернення каретки) та #10 (переведення рядка). Наприкінці файла записується *маркер кінця файла* - керуючий символ #26. Приклад текстового файла, що складається з двох рядків, наведено на рис. 9.1.

1-й рядок файла	Т	Н	К	С	т	0	в	И	Й	#13	#10
2-й рядок файла	Ф	А	Й	л	#13	#10	#26				

Рис. 9.1. Текстовий файл, що є сукупністю рядків

Бінарні файли в мові Разсаі поділяються на *типізовані* та *нетипізовані.* Типізований файл складається з компонентів одного типу. Кожний компонент має порядковий номер, номер першого компонента дорівнює нулю. Інформація в типізованих файлах зображується в тому самому вигляді, що і в пам'яті комп'ютера, і тому відпадає потреба у використанні керуючих символів типу кінця рядка або повернення каретки. Приклад зображення даних у файлі, що містить компоненти рядкового типу, наведено на рис. 9.2.

1-й компонент	#9	Т	Е	К	С	Т	0	в	и	й	#0
2-й компонент	#4	ф	А	Й	л	#0	#0	#0	#0	#0	#26

Рис. 9.2. Текстовий файл, що містить компоненти рядкового типу

Нетипізований файл у мові Разсаі розглядається як сукупність байтів. Компонентом нетипізованого файла вважається запис, довжина якого за замовчуванням становить 128 байт.

Класифікуймо файли за методом доступу. Файли послідовного доступу забезпечують доступ до поточного компонента тільки після вибору попереднього, а файли прямого доступу забезпечують вибір компонента за його номером. Текстові файли можуть бути лише файлами з послідовним доступом. До компонентів бінарних файлів можливий прямий доступ.

Синтаксис оголошення *файлової змінної* залежить від типу файла. Далі наведено синтаксис оголошення змінної текстового, типізованого та нетипізованого файла.

```
yar <ім'я файлової змінної>: Texi;
yar <ім'я файлової змінної>: Ti 1e oT <тип компонентів>:
yar <іи'я файлової змінної>: Tiie;
```

Приклади оголошення файлових змінних наведено нижче.

Приклад 9.1 _ _ _ _ _

```
рrодгат ex9_1:
Type
  Oaïa=recor(i
    пате:$1:гіпд;
    асісігезз :5i:гіпд;
    ЬігШауііпїедег;
  epci;
yar TI :Tex£;                    {текстовий файл     }
    Ьіпі;Tiie oT cIaTa;          {файл записів       }
    Ьіп2:Tiie oT inTедег         {файл цілих чисел   }
    пТ: Ti 1 e;                  {нетипізований файл }
Ьеділ
epci.
```

УВАГА

Якщо формальний параметр підпрограми має файловий тип, то він має бути параметром-змінною. Передача файлової змінної до підпрограми як параметра-значення спричинить синтаксичну помилку, оскільки значення цієї змінної не можна скопіювати у стек.

9.2.2, Установка відповідності між фізичним і логічним файлами

Файл будь-якого типу може бути оброблений у програмі лише після того, як певна файлова змінна буде зв'язана з певним фізичним файлом. Це зв'язування виконується за допомогою процедури Аззідп, що має такий синтаксис:

```
Аззідп(<файлова змінна>,<рядковий вираз>):
```

Значенням рядкового виразу має бути ім'я фізичного файла. Формат цього імені визначається операційною системою. Нагадаємо, що повне ім'я файла в операційних системах МЗ-Б05 та \¥іпс1о\У5 має такий вигляд:

<ім'я логічного диска>:\<ім'я каталогу 1>\<ім'я каталогу 2>\...\ **<ім'я** файла**>.<розширення>**

Файл має знаходитися в робочому каталозі програми, якщо в його імені не вказано шлях до нього. Якщо ім'я файла задається порожнім рядком, то файлова змінна зв'язується зі стандартними файлами і приТ та оиТриТ, про які йтиметься нижче.

Виклик процедури Аззідп має передувати викликам усіх інших процедур обробки файлів. У прикладі 9.2 виконано зв'язування файлових змінних із фізичними файлами.

Приклад 9.2

```
ргодгат ех9_2;
Type
    ciaTatecorcI
      паше:зїгіпд;
      асісігезз :зІгіпд;
      Ьі гТИсіау: і пїедєг:
    епсі;
уаг £1:IexI;
    Ьіпі: Ті1є оТ cIaTa;
    Ьіп2:Ті1е оТ іпїедег;
    пі: Тії е;
Ьеділ
  Аззідп(11, 'cIоситепТ.ІхГ);
  Аззідп(Ьіп1,'апкєіа.сіос');
  Аззідп(Ьіп2." сі:\Ьр\питЬегз. cIaT');
  АззідпСпі,'ЬуТесЬаіп.Ьіі'):
епсі.
```

9.2.3. Відкриття та закриття файлів

Фізичний файл, із яким була зв'язана файлова змінна, може перебувати в одному з таких станів.

- Файл відкритий для читання: читання даних із файла дозволено, а запис до файла — заборонено.
- Файл відкритий для запису: дозволений лише запис даних до файла.
- Файл відкритий для читання і запису: дозволено як читання, так і запис до файла. У цьому стані не можуть перебувати текстові файли.

4- Файл закритий: із файлом не можна виконувати жодних дій.

Операції відкриття та закриття файлів виконуються відповідними процедурами, які розглянемо детальніше.

Відкриття файла для читання або для читання і запису виконується процедурою КєзеТ, для якої є два варіанти синтаксису:

```
Кезе1(<файлова змінна>);
КезеЦуаг <файлова змінна> : Piie; <розмір запису> : НогсI);
```

Перший варіант процедури використовується для текстових і типізованих файлів, другий — для нетипізованих. Текстові файли процедура КезеТ відкриває лише для читання, а бінарні файли - для читання і запису.

Параметр <розмір запису> використовується під час обміну даними з нетипізованим файлом. Розмір запису нетипізованого файла впливає на швидкість обміну даними між диском і пам'яттю. Найбільшу швидкість можна забезпечити, якщо задати розмір запису рівним розміру кластера. Нагадаємо, що кластер складається з фіксованого числа секторів дискового носія. Кластер може бути прочитаний або записаний протягом одного звернення до диска.

Із кожним відкритим файлом зв'язаний *файловий покажчик*, що вказує на той компонент файла, над яким буде виконано наступну операцію зчитування або запису. У разі виконання такої операції файловий покажчик зсувається на наступний компонент. Під час відкриття файла файловий покажчик встановлюється на початок файла, тобто на компонент із порядковим номером 0.

В результаті спроби відкрити файл, якого немає на диску, виникне помилка **Error #2: File** поі ТоипсІ (Файл не знайдено). Щоб запобігти перериванню програми внаслідок спроби відкрити неіснуючий файл, використовують директиву компілятора {$1-}. Вона вимикає автоматичний контроль помилок введення та виведення. Директиву {$1+} використовують для ввімкнення такого контролю. Якщо контроль помилок введення-виведення вимкнено, то для перевірки наявності файла на диску можна використовувати функцію **IOKezиH.** Коли файл існує на диску, функція ІОКезиІТ повертає значення 0. Приклад 9.3 показує, як використовуються директиви компілятора **{$1-},** {$1+} і функція **IOK**єБ**Л**І: під час відкриття файла.

Приклад 9.3

```
ргодгат ех9_3;
уаг Р: Тіїе оТ ВуТе;
Ьеді п
  АззідпСР, 'Тіїе.ciai:[1]');
  {$1-}          {відключення контролю введення-виведення}
  КезеТ(Р);
  {$1+}          {включення контролю введення-виведення  }
  іТ ІОКезиН = 0 ТЬеп
    мгіТе1п('Рі1е із ореп')
  еізе
    мгіїеІпСРіІе поі Тоипсl');
епсі.
```

Відкриття файла для запису або читання і запису здійснюється процедурою Кемгіїе, для якої існує два варіанти синтаксису:

```
Кемгіїе(<файлова змінна>);
Кемгіі:е(уаг --файлова змінна> . Ті1е; <розмір запису> : Morci);
```

Перший варіант процедури **Кемгіїе** використовується для текстових і типізованих файлів, другий - для нетипізованих. Параметрами процедури є файлова змінна та розмір запису нетипізованого файла. Текстові файли процедура Rewrite відкриває лише для запису, а бінарні -- для читання і запису.

Якщо аргумент процедури Кемгіїе зв'язаний з іменем неіснуючого файла, то файл з таким іменем буде створений. Якщо цю процедуру використати для відкриття вже наявного файла, то вміст файла буде видалено та створено новий порожній файл. Процедура Кемгі Ie встановлює файловий покажчик на початок файла.

Для дописування рядків до вже наявного текстового файла його потрібно відкрити за допомогою процедури Аррепсі:

Аррепсі(<файлова змінна>);

Після виклику цієї процедури файл стає доступним тільки для запису, а файловий покажчик встановлюється в кінець файла. Отже, всі рядки зберігаються, а нові символи дописуватимуться після наявного тексту.

Файли закриваються процедурою Сіозе:

```
С1озе(<файлова змінна>);
```

Процедура СІ озе забезпечує збереження всіх компонентів логічного файла у фізичному файлі. Якщо в логічний файл були записані певні дані, але його не було закрито, запис у відповідний фізичний файл міг здійснитися не повністю. Після закриття файла зв'язок файлової змінної із фізичним файлом не порушується, і файл повторно може відкриватися без додаткового виклику процедури Аззідп.

9.2.4. Зчитування і запис текстових файлів

Нагадаємо, що текстовий файл є сукупністю рядків змінної довжини, до яких можливий лише послідовний доступ. Як було зазначено вище, кожен рядок текстового файла завершується маркером кінця рядка, що складається із двох символів: #13 та #10. Кінець файла позначається символом #26. Ідентифікатором типу текстового файла є слово Iехї.

Зчитування даних із текстового файла здійснюється процедурами Кеасі та КеасП п за таким синтаксисом:

```
Кеасі(<файлова змінна>:<список введення>);
Кеасі1п(<файлова зм1нна>;<список введення>);
```

Тут **<список введення>** є переліком змінних символьного, рядкового, цілочислового або дійсного типу.

Після зчитування певного компонента файловий покажчик зсувається до наступного компонента. Якщо виконується зчитування з файла до змінної типу **сіїаг,** то процедура **Кеасі** зчитує один символ. Коли досягнуто кінця рядка, результатом зчитування є символ кінця рядка, #13, а коли досягнуто кінця файла, зчитується символ кінця файла, #26.

Під час зчитування значення до змінної цілочислового або дійсного типу спочатку виділяється підрядок, в якому видалені всі ведучі пробіли, символи табуляції (#9) і маркери кінця рядка. Далі зчитуються всі символи, що утворюють число зі знаком. Зчитування припиняється при виявленні першого пробілу, маркера кінця рядка або символу табуляції. Зчитаний рядок цифрових символів перетворюється на число, що присвоюється відповідній змінній. Наступна операція зчитування починається з пробілу (маркера кінця рядка, символу табуляції тощо).

Якщо виділений підрядок містить нецифрові символи, то виникає помилка введення-виведення Error 106: **Invalid numeric format** (Некоректний числовий формат).

Під час зчитування даних до змінної типу string процедура Readln зчитує всі символи до маркера кінця рядка. Якщо кількість символів рядка у файлі більша за кількість, вказану в оголошенні рядкової змінної, то зайві символи не зчитуються. Якщо процедурою Readln певний рядок було зчитано повністю, то під час наступного її виклику зчитування починається з маркера кінця рядка, а значить, буде зчитано рядок нульової довжини. Отже, процедура Readln не переводить файловий покажчик на наступний рядок символів і тому її не можна використовувати для зчитування послідовності рядків.

Приклад 9.4

Розглянемо приклад зчитування даних із текстового файла у змінні різних типів. Результат роботи програми зображено на рис. 9.3. Програма опрацьовувала файл, який містив такі дані: 5 -20 1 2 3 4 5 зігіпд.

```
program ex9_4;
var T:TextT;
    a,i :integer;
    b:real;
    s:string;
    mas:array[1..5]of integer;
begin
  writeln('Reading from the text file');
  Assign (T,'T.TxT');
  Reset(T);
  readln(a,b);
  for i:=1 to 5 do
    readln(T,mas[i]);
  readln(s);
  writeln('a=',a,' b=',b,' s=',s);
  writeln('array of integers:');
  for i:=1 to 5 do
    write(mas[i],' ');
  writeln;
  Close(T);
end.
```

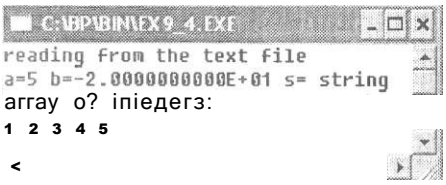


Рис 9.3. Результати роботи програми ex9_4. Зчитування даних із текстового файла

Процедура КеасЛп зчитує всі символи рядка із символом його кінця включно, що забезпечує переведення файлового покажчика на новий рядок. Процедуру можна викликати без параметрів, що спричинить переведення файлового покажчика на початок наступного рядка файла без зчитування попереднього рядка. Отже, для введення даних із одного рядка файла використовують процедуру Keaci, для введення даних із різних рядків - процедуру **KeacЛп**. Застосовуючи процедуру КеасЛп для зчитування чисел, слід враховувати, що після зчитування останньої цифри числа всі символи тексту, що залишилися до маркера кінця рядка, будуть пропущені, і доступним стане перший символ наступного рядка текстового файла.

Приклад 9.5_

Наведемо програму, що виводить на екран перші п'ять рядків текстового файла.

```
ргодгат ех9_5;
уаг Т:ТехТ;
    з:5Їгіпд;
    і:іпТедег;
Ьеді п
  мгііе!п('геасііпд го^з Тгот ТНе ТехТ Tiïe');
  A$$ідп (Т,'Т.ТхТ');
  Кезеї(Т);
  Тог і:=1 То 5 сіо
  Ьеді п
    геасі1п(Т,5);
    НГІТеІп(з);
  епсі;
  СІозе(Т);
  геасії n;
епсі.
```

Запис до текстового файла здійснюється за допомогою процедур МгіТе та Мпі - Теїп, що мають такий синтаксис:

```
МгіТе(<файлова змінна>:<список виведення>);
МгіТе1п(<файлова змінна>:<список виведення>);
```

Тут <файлова змінна> - змінна типу ТехТ; **<список виведення>** - перелік змінних або виразів символьного, рядкового, цілочислового, дійсного чи логічного типу.

Зауважимо, що процедура запису до текстового файла відрізняється від процедури виведення на екран лише тим, що першим її параметром є файлова змінна. Тому всі особливості застосування процедур МгіТе і МгіТеїп, що були розглянуті в розділі 2.5.3, зберігаються і в разі виведення даних до текстового файла.

Різниця між процедурами ИгіТе та **MriTeIn** полягає в тому, що рядок, який записується до файла за допомогою процедури **MriTeIn,** завершується символом кінця рядка. Якщо процедура НгіТеїп використовується без списку виведення, то до файла записується порожній рядок.

Процес запису до текстового файла даних різних типів проілюстрований у наведеному далі прикладі 9.6.

Приклад 9.6

```
ргодгат ех9_6:
yar T: "Ьехі;
    з:зТппд;
    а,і:іпТедег;
    Ь:геа1;
Ьеді п
  мгіТе1п( 'IIзаде оТ THe ргосесііге мгіТеїп То Тіїе');
  Аззідп (Т.'ехатрІе.ТхТ');
  Ремгіїе(Т);
  Тог і :=1 То 5 сіо
  Ьеді п
    мгіТе1п('епТег іпТедег,геаі, зТгіпд'):
    геасЛп(а,Ь,$);
    мгіТеїпСТ,'а-'.а,' Ь='.Ь.' $='.з):
  епсі;
  СІОЗЄ(Т);
  геасії п:
епсі.
```

Вміст текстового файла **example.TxT** може бути, зокрема, таким:

```
а=1 Ь= 2.0000000000E+00 з= Разсаі
а=2 Ь= 3.0000000000E+00 з= С++
а=3 Ь= 4.0000000000E+00 з= Оауа
а=-4 Ь=-3.4500000000E+00 з= PHP
а=5 Ь= 6.7800000000E+00 з= ИТтІ
```

Під час зчитування з текстового файла кількість його рядків зазвичай є невідомою. Тому для зчитування з файла всіх даних потрібно використовувати функцію **ЕоТ,** яка визначає, чи досягнуто кінця файла. Наведемо синтаксис цієї функції.

```
ЕоТ(<файлова змінна>) : Ьооїеап;
```

Дана функція повертає значення булевого типу. Якщо файловий покажчик посилається на кінець файла, буде повернено значення Тгие. Отже, послідовне зчитування всіх компонентів із файла, розмір якого невідомий, може бути реалізоване таким циклом:

```
иьііе поТ еоТ(Т) сіо
   КеасЛп(Т.з);
```

Функція Еоіп визначає, чи посилається файловий покажчик на маркер кінця рядка. Наведемо її синтаксис:

```
Ео1п(<файлова змінна>) : Ьооїеап;
```

Якщо поточним символом є маркер кінця рядка, функція **Еоі п** повертає значення Тгие.

Приклад 9.7

Розв'яжемо таку задачу. Потрібно створити текстовий файл шляхом введення його рядків з клавіатури, а у кожному рядку створеного файла знайти найдовше

слово і дописати його в кінець рядка. Вважаємо, що слова відокремлюються одне від одного довільною кількістю пробілів.

Розробку програми почнемо із визначення глобальних змінних. Файлові змінні слід оголошувати як глобальні. Це забезпечить можливість їх використання у будь-яких підпрограмах. Отже, для файла, що створюється під час введення рядків із клавіатури, оголосимо змінну TI, для файла, що доповнюється найдовшими словами, — змінну T2. Оскільки за умовою розглядаються текстові файли, то файлові змінні належатимуть до типу TexT,

Алгоритм програми пошуку найдовших слів у рядках текстового файла

1. Зв'язати файлові змінні з фізичними файлами.
2. Створити текстовий файл шляхом введення його рядків з клавіатури та вивести його для контролю.
3. Обробити послідовно рядки файла.
   3.1. Зчитати рядок із файла у рядкову змінну
   3.2. Визначити у рядку найдовше слово та дописати його в кінець рядка.
   3.3. Отриманий рядок записати у новий файл.
4. Вивести файл, який було створено в результаті виконання кроку 3.

Розробимо процедури, що виконуватимуть кроки 2-4 даного алгоритму. Процедура створення вхідного файла CreaTe_Ti відкриватиме файл TI для запису і дозволить користувачеві вводити рядки до нього доти, доки користувач не дасть негативної відповіді на запит conTinue ? [y/n] (продовжувати? [так/ні]).

Алгоритм процедури Ac1c!\\Tor(ї, що визначає найдовше слово у рядку

1. Зчитати рядок із файла у рядкову змінну.
2. Скопіювати зчитаний рядок для подальших перетворень.
3. Вважати перше слово найдовшим. Присвоїти його змінній iorci, а його довжину — змінній max!.
4. Повторювати дії, зазначені у кроках 5- 7, доти, доки рядок не стане порожнім.
5. Видалити з рядка перпіе слово.
6. Знайти перший пробіл у рядку, визначити кількість символів до нього та вважати це значення довжиною шуканого слова. Якщо першим символом рядка є пробіл, то шукане слово порожнє, а його довжина дорівнює нулю.
7. Порівняти обчислену на кроці 5 довжину слова із значенням змінної шахі. Якщо значення тахі виявилося меншим, присвоїти змінній тахі обчислену довжину слова, а саме слово присвоїти змінній morci.
8. Дописати слово, що міститься у змінній morci, наприкінці копії вхідного рядка, що її було створено під час виконання кроку 2.

Процедура OuT виводить файл, зв'язаний з її параметром. Принцип роботи цієї процедури є очевидним.

```
ргодгат ex9_7;
изез crT;
yar
   T1,T2:їexT;
```

```
{========== створення текстового файла
procegiure CreaTe_TI;
V ar
    key :cipar;          {ознака завершення введення даних}
    з:зTripd;            {рядок, у який вводять дані      }
Ьедіп
  KemriTe(TI);                  {відкрити файл для запису }
  repeaT
    »гПе1п( 'іприі; зTripd;');
    reaain(з);                  {ввести з клавіатури рядок}
    мгiTeIn(TI.з);              'записати рядок у файл     }
    мгiTe1n('conTinue ? [y/n]');
    key :=reac!key;
  unTii ирсазе(key)='N':
  Clozе(TI);
epci;
{======== додавання до рядка найдовшого слова
procegiure Acicikiorci:
yar
  тахі:inTедег;             {довжина найдовшого слова    }
  IьГОГСІ,                  {найдовше слово у рядку      }
  за,зЬ;зTripd;             {зчитаний і записаний рядки  }
Ьедіп
  KЄБЄT(TI);                {відкрити файл для читання   }
  KemriTe(T2);              (відкрити файл для запису    }
  мЬііе поТ eoT(TI) cio     {доки не кінець файла        }
  Ьедіп
    reaciIniШ.за);          {читати рядок із файла       }
    за:=за+' ';             {додати пробіл у кінець рядка}
                            {скопіювати рядок            }
    ІЇМХІ ; =роз (' ,за)-1; {перше слово                 }
    иогсі ;=copy(за, 1,тахі! ; {вважати найдовшим        }
    міліє роз( за)<>0 cio {доки у рядку є пробіли    }
    Ьедіп
      cie1eTe(за,1,роз(' '.за)); {видалити перше слово  }
    {якщо максимальна довжина менша довжини поточного слова.
      iTшахі<роз(' ',5а)-1ТИеп
      Ьедіп     {то запам'ятати нову довжину та слово}
        тахі :=роз(' ',за)-1;   {найбільша довжина слова}
        могсі;=copy(за,I.тахі); {найдовше слово        }
      epci;
    epci;
    за;=conca T(зЬ,могсІ); {дописати слово у кінець рядка}
    мгiTe1n(T2,за);         {записати рядок у новий файл  }
  epci;                     {кінець циклу читання файла   }
  Clozе(TI);
  C1ozе(T2),
epci;
(========== виведення ( іаила на екран ========
procegiure OuT(yar TexTTI' є;TexT);
yar  5:$Tripd;              {рядок, зчитаний із  аила
Ьедіп
   KezeT(TexTTile);         (відкрити файл для читання,
```

```
   miiiie noT eoT(ïexTTiIe) cio {доки не кінець файла     }
   Ьеділ
     reaciïnCTexTTiïe,з):      {читати рядок файла       }
     мгiTeIп(з);               {вивести рядок на екран   }
   епсі;
   CIозе(TexTTiIe);
епсі;
Ьеділ
  Аззідп(Т1,'ex9_7.TxT');     {зв'язати логічні         }
  А55ідп(Т2.'ex9_7п.TxT');    {та фізичні файли         }
  CreaTe_T1;                                 _____
  мгiTeIпC      зоигсе Ti1e ex9_7.TxT      ');
  Оиї(TI);
  AcicMorci; _____                           _____
  мгiTeїпГ      гезиIT Ti1e ex9_7п.їxT     ');
  0иТ(Т2);
  reaciї п;
епсі.
```

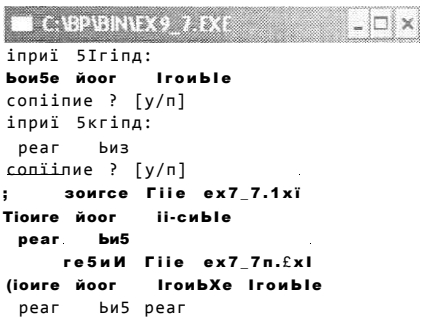


РИС. 9.4. Результати роботи програми ex9_7. Створення та доповнення текстового файла

У мові Разсаі означені стандартні текстові файли і прИТ та оиТриТ. Вони вважаються відкритими під час виконання операцій введення з клавіатури та виведення на екран, тому процедури мп'Те, мгiTeIп, reacі і reacії п за замовчуванням використовують файлові змінні іприТ та оиТриї. Наприклад, еквівалентними будуть такі оператори:

```
мгiTe1п('оиТриТ To TЬe зсreeп');
мгiTe1п(оиТриТ.'оиТриї To Тїе зсreeп');

reaciIп(іприТ.a.b);
reac11п(a,b);
```

Текст можна виводити не тільки на екран або диск, але і на інші стандартні пристрої виведення, такі як комунікаційні порти введення-виведення, принтер або порожній (ШИ.) пристрій. З метою моделювання процесу виведення на ці пристрої використовуються текстові файли з іменами ССМ, РКИ, [_РТ1 тощо. Логічне

ім'я CON відповідає так званій консолі, введення даних через яку здійснюється з клавіатури, а виведення — на екран. Логічні імена LPT1, LPT2,1_PT3 зіставлені з паралельними портами, через які здійснюється виведення текстів на принтери. Якщо до комп'ютера підключено один принтер, то найчастіше використовують ім'я LPT1 або його синонім PRN Для передачі даних із комп'ютера на комп'ютер та з метою керування мишею використовують послідовні порти з логічними іменами COM1 та COM2. Синонімом імені COM1 є ім'я AUX. Логічне ім'я NUL означає порожній пристрій, що інтерпретується як приймач інформації, що має необмежену ємність. Цей пристрій використовують під час налагодження програми.

9.2.5, Послідовний запис і зчитування компонентів бінарних файлів

Нагадаємо, що в мові Разсаі визначено два різновиди бінарних файлів: типізовані і нетипізовані файли. Типізований файл є послідовністю однотипних компонентів. Нетипізований файл можна відкрити лише як файл компонентів однакового розміру, тобто як файл однотипних компонентів. Тому надалі, в розділах 9.2.5 і 9.2.6, типізовані і нетипізовані бінарні файли не розрізнятимемо. Компоненти бінарного файла нумеруються, але їх кількість не є наперед відомою, як у масивах. Між компонентами не записуються жодні роздільники. Такого поняття, як рядок, а отже, і маркер кінця рядка, для бінарних файлів не означено.

Зчитування з бінарних файлів здійснюється лише процедурою ReadL, а запис — лише процедурою WriTe. Застосування до бінарних файлів процедур ReadLn або WriteLn є неприпустимим, оскільки для таких файлів не означено поняття рядка. Синтаксис виклику процедур зчитування і запису у бінарні файли є таким:

```
ГСеасК<файло8а змінна>;<список введення>);
кІгіі:е(<файлова змінна>;<список виведення>);
```

Тут <список введення>і <список виведення> - це перелік змінних того самого типу, що й тип компонентів файла.

УВАГА

Запис констант до бінарного файла є неприпустимим. Наприклад, для того щоб до файла записати значення 1, потрібно це значення присвоїти змінній, ім'я якої вказане у списку виведення процедури \Л/гііе.

Зазначимо, що один і той самий фізичний файл може бути зіставлений із різнотипними логічними бінарними файлами. Тому можна зчитувати значення компонентів того самого файла у змінні різного типу. Як використовується така технологія продемонстровано у прикладі 9.8.

Приклад 9.8

Створимо бінарний файл із компонентами типу Ьуїе і відобразимо його вміст двома способами: як послідовність символів і як послідовність чисел, які є ASCII-кодами відповідних символів.

Оголосимо дві файлові змінні: T — файл із компонентами типу ЬуТе і TT — файл із компонентами типу сііаг. Процедурою Аззідп зв'яжемо ці файлові змінні з одним і тим самим фізичним файлом Рігзї.ciaT. Спочатку дані у РігзТ.ciaT введемо через змінну Т (процедура СгеаТе). Потім, використовуючи цю саму змінну Т, відобразимо вміст файла як послідовність чисел і закриємо файл (ці дії виконує процедура ЗіюмВуТе). Відкриємо фізичний файл РігзТ.cial: повторно як логічний файл TT і відобразимо його вміст як послідовність символів (процедура ЗИомСНаг).

```
ргодгат ех9_8;
уаг
  Т:Ті1е оТ ЬуТе;   {файл байтів                          }
  TT:Ті1е оТ сііаг; {файл символів                        }
  а:ЬуТе:           {змінна для читання чисел із файла    }
  сЬ;сЬаг;          {змінна для читання символів із файла}

процесіиге СгеаТе:
уаг і:іпТедег;      {параметр циклу                       }
Ьеді п
  КемгіТе(Т);       {відкрити файл для запису             }
  мгіТе!п('іприТ 10 іпТедегз');
  Тог і :=1 То 10 сіо
  Ьеді п
    геасі(а);       {ввести число з клавіатури            }
    мгіїе(Т.а);     {записати число у файл                }
  епсі:
епсі;
{======= виведення файла як послідовності чисел ========}
процесіиге ЗИомВуТе:
Ьеді п
  КезеТ(Т);             {відкрити файл для читання        }
  мНііе поТ еоТ(т) сіо  {поки не досягнуто кінця файла,   }
  Ьеді п
    геасКТ.а):      {зчитати компонент у змінну типу ЬуТе}
    мгіТе(а,'  ');       {вивести значення змінної        }
  епсі;
  СІозе(Т);             {закрити файл                     }
  мгіТеІп;
епсі;
{=—==== виведення файла як послідовності символів ======}
процесіиге ЗЬомСИаг;
Ьеді п
  КезеТ(ТТ);
  мЬіІе поТ еоТ(тТ) сіо {доки не досягнуто кінця файла,  }
  Ьеді п
    геасКТТ.сіі);   {зчитати компонент у змінну типу сіїаг}
    мгіТе(сИ,'  ');     {вивести значення змінної        }
  епсі;
  СІозе(ТТ);
  мгіТеїп;
епсі;
```

```
Ьеділ
  Аззідп(Т,'Рігзї.сіаТ');
  Аззідп(ТТ,'Рігзь.сіат.');
  СгеаТе;
  игіТе1п('епТегесі питЬегз Тгот РігзТ.сіаТ');
  ЗНомВуТе;
  ЗНомСїїаг;
  геасії п;
епсі.
```

```
іприЬ 1В In1едег5
1 2 3 4 5 6 7 8 9 10
епіегей питЬегз Ягогі Рії-ЕЇ.йаІ
1 2 3 4 5 6 7 8 9 10
```

РИС. 9.5. Результати роботи програми ех9_8. Послідовне зчитування та запис бінарних файлів

9.2.6. Прямий доступ до компонентів бінарних файлів

Компоненти бінарного файла мають однаковий розмір. Цим зумовлені певні переваги бінарного файла над текстовим. По-перше, стає можливим прямий доступ до компонентів файла за їхніми номерами. Під прямим доступом розуміється можливість встановлювати файловий покажчик на заданий компонент без послідовного перебирання попередніх компонентів. По-друге, відкриття бінарного файла процедурою КезеТ робить можливим як читання, так і запис довільних компонентів файла без руйнації його вмісту. Це пояснюється тим, що модифікація кількох компонентів усередині бінарного файла не призведе до зміни їх розміру, а отже, не потребуватиме зсуву інших компонентів. Нагадаємо, що процедура КемгіТе також відкриває бінарний файл у режимі читання і запису, але при цьому вона знищує його вміст (якщо файл уже існував).

Прямий доступ до компонентів бінарного файла в мові Разсаі здійснюється за допомогою процедур та функцій, наведених у табл. 9.1.

Таблиця 9.1. Процедури та функції прямого доступу до компонентів бінарних файлів

Ім'я функції або процедури	Призначення
5еек(уаг Т; п: ЬопдіпТ);, процедура	Переміщує файловий покажчик на компонент з номером п. Файл Т має бути відкритий
ТгипсаТе(уаг Т);, процедура	Видаляє всі компоненти файла Т від поточного компонента до кінця файла
Рі1еРоз(уаг Т): ІопдіпТ;, функція	Повертає номер запису, на який посилається файловий покажчик
Рі1е$І2е(уаг Т): !_опдіпї;, функція	Повертає кількість компонентів файла Т

Механізм використання наведених у табл. 9.1 процедур і функцій проілюстровано у прикладі 9.9.

Приклад 9.9

Розв'яжемо задачу сортування вмісту файла. Компонентами файла будуть записи з інформацією про працівників певного підприємства:

```
Type рєгзоп=гєсогсі
       пате:зТгіпд:     {ім'я працівника }
       заіагу:reai:     {оклад           }
     епсі:
```

Відсортуємо файл бульбашковим методом за зростанням окладів працівників. Під час сортування компоненти файла потрібно буде переставляти. Для збереження у пам'яті значень компонентів, що переставляються, оголосимо глобальні змінні сотрі і сотр2 типу регзоп. Ці ж самі змінні можна використовувати і під час введення або виведення вмісту файла. Сам фізичний файл T.ciaï зв'яжемо із файловою змінною Т.

Зчитування даних з клавіатури та їх запис у файл здійснюватимуться у процедурі CreaTe. Значення полів запису **регзоп** слід вводити окремими операторами **"eacii п,** але записувати дані про працівника у файл треба одним оператором мгіТе:

```
  КеигіТе(Т);          {відкрити файл для запису            }
  Тог і:=1 То п сіо    {цикл введення значень 10 компонентів}
  Ьеділ
    геасіїп(сошр1.пате);          {ввести дані з клавіатури}
    геасії п(сотр1. за! агу):
    мгіТе(Т,сотрі);               {записати дані у файл     }
  епсі;
```

Вміст файла виводитимемо процедурою ОиТ, що викликатиметься після створення файла та його сортування. Принцип дії цієї процедури є очевидним.

Під час сортування файла використаємо засоби прямого доступу до його компонентів. Спочатку встановимо файловий покажчик на початок файла: зеек(Т.О). Після цього зчитуватимемо з файла сусідні компоненти та порівнюватимемо значення їх полів заіагу. Якщо компоненти не впорядковані, їх слід переставити:

```
зєек(Т,;і);                  {встановити файловий покажчик
                               на компонент з номером ^       }
геасі(Т,сотр1,сотр2):       {читати два сусідніх компоненти  }
іТ сотрі.за1агу>сотр2.заіагу Тїеп
Ьеділ                        {якщо компоненти не впорядковані,}
  зеек(Т.з);                 {встановити файловий покажчик
                               на компонент з номером ]
                               та записати компоненти в файл  }
  «гіТе(Т,сотр2,сотрі);     {у зворотному порядку            }
епсі;
```

Повний текст програми сортування файла наведено нижче. Результати виконання програми подано на рис. 9.6.

```
ргодгат ех9_9;                  {сортування бінарного файла}
Type регзоп=гесогсі             {тип запису працівника     }
       паше:зТгіпд;             {ім'я працівника           }
       за1агу:rea1;             {заробітна плата           }
     епсі;
```

```
yar Г.ТЛе оТ регзоп:            {файл записів              }
    сотрі,сотр2:регзоп;         {компоненти файла          }
    п: іпТедег;                 {кількість працівників     }
|================== створення файла ====================}
ргосесііиге СгеаТе;
yar і:іпіедег:              {параметр циклу                }
Ьеді п
  Ке«гіїе(Т);               {відкрити файл для запису даних}
  мгіТе1п('еппТег питЬег оТ регзопз');
  геасііп(п);
  Тог і :=1 То п сіо        {цикл введення даних у файл    }
  Ьеді п
    мгіТе('епТег пате: [1]);
    геасіїп(сошр1.пате);
    мгіТе('епТег заіагу: ');
    геасі1п(сотр1 .заіагу);
    мгіТе(Т,сотрі);         {записати дані у файл          }
  епсі;
епсі;
{================== сортування файла ==================}
ргосесііиге ЗогТ;
yar і. з: іпТедег;
Ьеді п
  зеек(Т.О);{встановити файловий покажчик на початок файла
            перебирати компоненти з кінця файла до початку}
  Тог і :=Ті1езіге(Т)-1 сіокпТо 1 сіо
          {перебирати компоненти від нульового до поточного}
    Тог ,] :=0 То і-1 сіо
    Ьеді п                  {впорядкувати сусідні компоненти}
      зеек(Т.,і);
      геасКТ,сотрі,сотр2);
      іТ сотрі.за1агу>сотр2.заіагу ТЬеп
      Ьеді п
        ЗЄЄк(Т.з);
        мгіТе(Т,сотр2.сотрі);
      епсі;
    епсі;
епсі;
{================ виведення вмісту файла — — — — = = }
ргосесііиге Оиї;
Ьеді п
  зеек(Т.О); {встановити файловий покажчик на початок файла}
  мЬіІе поТ еоТ(Т) сіо      {доки не досягнуто кінця файла,}
  Ьеді п
    геасі(Т,сотрі);         {зчитувати компоненти          }
    игіїе1п(сотр1.пате:8,сотрі.заіагу:10:2);{і виводити їх }
  епсі;
епсі;
|================== основна програма ====================}
Ьеді п
  мгіТе1п('зогїіпд оТ Тґіе Ьіпагу Тіїе');
  Аззідп(Т,' Т.сіаТ');
  СгеаТе;                   .,      {створити файл       }
  мгіТе1п( 'сгеаТесі Тіїе');
```

```
  игіІе1п('=== пате === заіагу
  Оиі;                              {вивести вміст файла     }
  5огІ;                             {відсортувати вміст файла}
  мгіІе1п( 'зогЬесі  Тіїе');
  мгіІе1п( '=== пате == заіагу
  ОіЛ;                              {вивести вміст файла     }
  СІозе(Т);
  геасії п;
епсі.
```

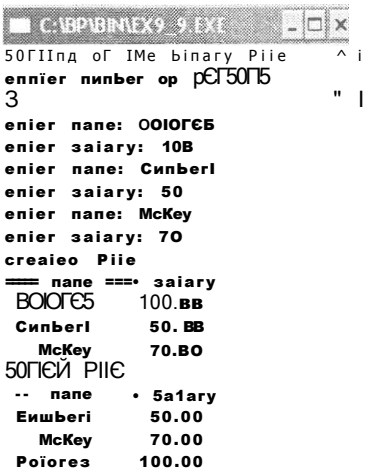


РИС. 9.6. Результати роботи програми ex9_9. Сортування бінарного файла

9.2.7. Системні операції з файлами

Крім процедур і функцій, розглянутих раніше, у мові Разсаі визначено процедури, за допомогою яких можна видалити файл або перейменувати його, тобто виконати операції, властиві операційній системі. Для видалення файла використовують процедуру Егазе, а для перейменування - процедуру Кепате. Синтаксис зазначених процедур є таким:

```
Егазе(<файлова змінна>);
:,епате(<файлова змінна>,<нове ім'я>);
```

Тут <нове і м' я> — рядок, який має задовольняти вимоги операційної системи щодо специфікації імен файлів. Під час виклику процедур Егазе та Кепаше файл має бути закритим. Як приклад використання цих процедур наведемо код програми, що перейменовує файл.

```
уаг Тіїехі;
Ьеді п
   Аззідп(Т, 'Т.їхі:');
   Кепате(Т,'асісігезз.сіаі');
епсі.
```

Процедури та функції, призначені для пошуку файлів на диску, керування атрибутами файла тощо, означені у бібліотечних модулях **dos** та міпсіоз.

9.3. Буферизація даних

Буфером називається область пам'яті, призначена для тимчасового збереження даних під час їх передачі від джерела до приймача інформації. Застосування буферів дає можливість зменшити диспропорції між швидкостями роботи процесора та зовнішніх пристроїв. Переважаюча частина зовнішніх пристроїв може отримувати та надсилати дані лише досить великими порціями (у сотні або тисячі байтів), і при цьому кожна операція обміну даними між пам'яттю і пристроєм є досить трудомісткою. Натомість, під час створення програм часто виникає потреба надіслати чи отримати з пристрою один або кілька байтів інформації, виконати певні дії, знову здійснити обмін даними із пристроєм тощо (див. програму сортування файла з прикладу 9.9). Було б украй недоцільно багаторазово повторювати трудомістку операцію обміну великою порцією даних із пристроєм, аби переслати один чи кілька байтів. Буфер дозволяє накопичити порцію даних у пам'яті, а потім передати її на пристрій «в один прийом». Отже, за допомогою буфера імітується обмін даними між програмою і зовнішнім пристроєм, в той час як насправді відбувається обмін даними між програмою та пам'яттю.

Під час зв'язування файлової змінної з фізичним файлом автоматично створюється файловий буфер. Кожному буферу відповідає покажчик, що посилається на його поточний елемент. Значення цього елемента присвоюється черговому файловому компоненту під час його запису. При зчитуванні даних із файла значення його чергового компонента копіюється в поточний елемент буфера. Покажчик буфера, як і файловий покажчик поточного компонента, у програмі явно не оголошується.

Розмір буфера визначається операційною системою і становить 2, 8, 16 або більше блоків. *Блок* є одиницею виміру обсягу даних під час обміну ними між диском та оперативною пам'яттю. Саме блоками дані копіюються з пам'яті на диск або з диску в пам'ять. Обсяг блока становить 512 байт, що є обсягом одного сектора диска.

Розглянемо принцип дії механізму буферизації детальніше. Під час завантаження операційної системи певна ділянка оперативної пам'яті резервується для виділення буферів. Ця ділянка називається *буферним пулом.* При зв'язуванні логічного та фізичного файлів операційній системі надсилається запит на відкриття каналу введення-виведення. У відповідь на цей запит операційна система виділяє буфер із буферного пула, і в нього зчитується певна кількість блоків даних із фізичного файла. Кожного разу після отримання запиту на зчитування дані вибираються з буфера та пересилаються в область пам'яті, в якій зберігаються значення змінних. Після зчитування останнього запису з буфера до нього копіюються нові блоки файла. При записі даних до файла відбувається зворотний процес: у буфері поступово накопичуються дані і коли буфер стає повним, він звільняється, а його вміст копіюється на диск. Якщо розмір фізичного файла не більший за роз-

мір буфера, то будь-яка кількість звернень до цього файла з боку програми потребує лише двох обмінів даними між буфером і диском, що відбуваються при відкритті та закритті файла. Під час закриття файла операційна система звільняє буфер, зв'язаний з каналом введення-виведення.

Мова Разсаі дає можливість програмісту створювати і використовувати власні буфери. Застосування цього механізму може суттєво підвищити швидкодію програми, що працює з файлами. Обмін даними через буфери, створені програмістом, можливий лише в разі використання нетипізованих файлів, що розглядатимуться в розділі 9.4.

9.4. Нетипізовані файли

Узагальненим файловим типом можна вважати нетипізований файл - файл, що розглядається як послідовність байтів. Довільний файл, створений як текстовий або типізований, можна відкрити та обробляти як нетипізований. Найважливішою характеристикою такого файла є розмір його запису. За замовчуванням він становить 128 байт, але програміст може задати й інший розмір. Нагадаємо синтаксис оголошення нетипізованого логічного файла:

уаг <ім**'я** файлової змінної>: **Тіїе;**

Відкриття нетипізованого файла для читання або запису виконується процедурами КезеТ та КемгіТе, яким крім файлової змінної передається, ще й додатковий аргумент — розмір файлового запису в байтах:

```
КезеТ(<файлова змінна>,<розмір запису>);
Ке«гіТе(<файлова змінна>.<розмір запису>);
```

Під час звернення до зовнішнього пристрою мінімально можливий обсяг даних, що передаються, становить 128 байт. Для забезпечення найвищої швидкості обміну даними між пам'яттю і диском слід встановлювати розмір запису рівним розміру кластера, що є кратним довжині фізичного сектора диска (512 байт).

Зчитування даних із фізичного файла у буфер здійснює процедура ВІоскКеасі, а запис даних з буфера у файл — процедура ВІоскМгіТе. Наведемо синтаксис виклику цих процедур:

```
В1оскКеасі(<файлова знінна>,<буферна змінна>,<кількість записів>[.<кількість фактично
зчитаних компонентів>]);

В1осШгіТе(<файлова змінна>,<буферна змінна>.«кількість записів>[,«кількість фактично
записаних компонентів>]);
```

Тут <файлова змінна> - ім'я змінної типу Тіїе; <буферна змінна> - ім'я змінної, що використовується як буфер обміну даними (ця змінна може мати будь-який тип); цілочисловий аргумент <кількість записів> визначає кількість записів, що мають бути зчитані або записані; необов'язковий параметр «кількість фактично записаних компоненті в> — змінна типу могсі, якій буде присвоєна кількість записів, переданих

насправді. Четвертий параметр слід використовувати тоді, коли кількість байтів, що їх залишилося передати, може бути меншою за значення третього аргументу.

Обсяг буфера можна визначити за такою формулою:

Обсяг буфера = кількість записів • розмір запису.

Тип змінної, що використовується як буфер введення-виведення, має бути таким, аби обсягу пам'яті, відведеної для збереження значень змінної, вистачило для розміщення всіх байтів під час читання чи запису даних. Обсяг даних, що передаються під час однієї операції обміну, не може перевищувати 64 Кбайт.

Приклад 9.10

Розв'яжемо задачу поділу файла на дві частини, розмір першої з яких задається користувачем. Для зв'язування логічних файлів із фізичними використаємо змінні Ttaiп, Tpar1;1 і Tpaï2 типу Tiïe. Роль буфера відіграватиме масив байтів ЬиTer, а його розмір визначатиметься константою сопзT. У змінних зігетаіп, 5іге1 та зі2е2 типу ІопдіпT зберігатимуться розміри вхідного файла та його частин.

Найпершою дією програми має бути зв'язування логічних файлів із фізичними. Виконавши зв'язування, один файл відкриємо для читання, два інших - для запису. Розмір вхідного файла визначимо за допомогою функції Tïïезіге. Виділення частини файла Ttaiп виконаємо у процедурі **ОІУРІІЄ**, що викликатиметься двічі. Оскільки процедура BlockKeacl, як і будь-яка інша процедура запису або зчитування компонентів файла, збільшує значення файлового покажчика на довжину зчитаної ділянки, то під час другого виклику процедури Оі **УРІ 1 Е** зчитування файла Ttaі п продовжиться з того байта, на якому воно припинилося під час першого виклику **01 УРІ 1 Е.**

Розглянемо процедуру Оі УРІ 1 е детальніше. Змінній кЬиTI присвоюється кількість повних блоків заданого розміру у певній частині вхідного файла, а змінній кЬиT2 присвоюється обсяг залишку частини вхідного файла, який менший за обсяг повного блоку. Зчитуючи блоки байтів із вхідного файла, записуватимемо їх у вихідний. Коли буде зчитано кЬиTI блоків, запишемо у файл залишок байтів.

```
ргодгат ех9_10;
сопзТ соипТ=512;                {розмір буфера
      гєсогсі5і2є=1:            {розмір запису, що зчитується
уаг Ttaiп,                          {вхідний файл
    TpagT1,TpagT2:Tiie;             {вихідні файли
    ЬиTer:array[1.,соипТ]оТ ЬуTe:   {буфер
    зігетаіп,                     {розмір вхідного файла
    5і2е1,зі2е2:ІопдіпТ:          {розміри вихідних файлів
{============ виділення частини файла ================
 ргосесіиге ОІУРІ1Є(УЗГ Т:Ті1е:уаг ехТепї:ІопдіпТ):
            {Т - файл, що утворюється, ехїепї - його розмір
 уаг  і.                {параметр циклу
      кЬиTI,            {кількість блоків
      кЬиТ2:«огсі;      {залишок байтів
```

```
Ьеділ
  кЬиТІ: =ехТепТ сііV соипТ;
  кЬиТ2: =ехїепТ шосі соипТ;
  Тог і:=1 То кЬиТІ сіо      {зчитати та записати блоки
  Ьеділ
    ВІоскКеасКТтаіп. ЬиТег, соипТ);
    В1оскИгі Те(Т.ЬиТег.соипТ);
    мгіТе1п('мгіТе Ьіоск =',і,' зіге оТ ЬІоск='.
             соипТ*recorcIзі2е);
  ЄПС;
  іТ кЬиТ<>20 ТНеп
  Ьеділ                {зчитати та записати залишок байтів}
    В1оскКеасК Тшаі п.ЬиТег.кЬиТ2);
    В1оскМгїТе(Т.ЬиТег,кЬиТ2);
    мг1Те1п( 'мгіТе гезісіиаі ЬуТез ='.кЬиТ2);
 епсі;
епсі;
{================= основна програма ====================}
Ьеділ
 мгіТе!п(' сіі V і сіє Тіїе іпТо 2 Тіїез');
 Аззідп(Ттаіп, 'Ттаіп.сіаТ');
 Аззідп(ТрагТ1,'ТП.сіаТ');                    ^
 Аззідп(ТрагТ2.'Т21.сІаТ');
 Кезеї(Ттаіп,recorcI5iie);      {відкрити вхідний файл    }
 КемгіТе(ТрагТ1,recorcізі2е);   {відкрити файли для запису}
 Ке«гіТеСТрагТ2,recorcізі2е);   {частин вхідного файла    }
 зі2етаіп;=Ті1езі2е(Ттаіп);     {розмір вхідного файла    }
 мгіТеїпС'зІ2ешаіп='.зігетаіп,   ЬуТез' );
 гереаТ                {вводити розмір першого  айла доти,}
   мгіТе!п('епТег зізе оТ Тіїе 1');
   геасПп(зігеІ);          доки не буде введене коректне}
 ипТЛ зіге1<=5І2етаіп;                          {значення}
 зі2е2:=зігетаіп-5і2е1;        {розмір другого файла     }
 «гіТе1п( 'creaTeci Тіїе 1:');
 0iyPi1e(ТрагТ1,зі2Є1);        {виділити першу частину   }
 игіТе1п('creaTeci Тіїе 2;');
 0iyPi1e(ТрагТ2.зіге2);        {виділити другу частину   }
 зігеІ^ТіІезігеСТрагТІ);    {розмір файла першої частини}
 мгіТе('creaTeci Тіїеі; ');
 мгїТєІп('зІ2єІ='.зігеї, ' ЬуТез' );
 5І2е2;=Ті1Є5І2Є(ТрагТ2);    {розмір файла другої частини}
 мгіТе('creaTeci Ті1е2: ');
 мгіТе1п('зі2е2=',зіге2. ' ЬуТез' );
 СІозе(Ттаіп);                  {закрити файли            }
 СІозе(ТрагТІ);
 С1озе(ТрагТ2);
 геасіїп;
епсі.
```

```
йічісіе ГІІЄ ІПІО 2 ГІІЄ5
5І2ЄГіаІП=1682 Ьуїез
епїег 5І5Є Ор РІІЄ 1
1000
creaieй Piie 1:
игіїе Ьіоск =1 зіге оР Ь1оск=512
игіїе rezirivai Ьуіез =488
creaieй Piie 2:
ЮГІІЄ ЬІОСк =1 5І2Є оР Ь1оск=512
игіїе rezійиai Ьуіез =170
creaїей Piiei: 5Ire1=1000 ЬуЬез
icreaieй Pi1e2: зire2=682 Ьуїез
```

M

Рис. 9.7. Результати роботи програми ex9_10. Поділ файла на дві частини

ВИСНОВКИ

4[i] Файл як фізичний об'єкт є іменованою областю на зовнішньому носії інформації. Розмір файла може бути довільним; обмежується він лише ємністю пристроїв зовнішньої пам'яті.

4 Файл як логічний об'єкт є послідовністю значень певного типу.

- Компоненти файла можуть належати до будь-якого типу, крім файлового. Кількість компонентів файла визначається під час роботи програми.
- Файли класифікують за типом компонентів і методом доступу до них. За типом компонентів файли поділяються на текстові та бінарні. За методом доступу розрізняють файли послідовного та прямого доступу.

4 Текстовий файл є сукупністю рядків змінної довжини. Кожен рядок завершується маркером кінця рядка — спеціальною парою керуючих символів #13 і #10. Наприкінці файла записується маркер кінця файла — керуючий символ #26.

4- Бінарні файли поділяються на типізовані та нетипізовані. Типізований файл складається з компонентів певного визначеного типу. Кожен компонент має порядковий номер, номер першого компонента дорівнює нулю. Нетипізований файл розглядається як сукупність байтів. Компонентом нетипізованого файла вважається запис, довжина якого за замовчуванням становить 128 байт.

4 Робота з файлом у мові Разсаі складається з таких етапів: оголошення файлової змінної; зв'язування файлової змінної з іменем існуючого файла або файла, що створюється; відкриття файла; обробка файла; закриття файла.

4 Зв'язування фізичного та логічного файлів здійснює процедура Аззідп, яку застосовують до закритих файлів.

- Файл для читання або для читання і запису відкривається процедурою **Резет.** Текстові файли процедура **Кезеї** відкриває лише для читання, **а** бінарні — для читання і запису.
- Файл для запису або для читання і запису відкривається процедурою **Кемпїіе.** Текстові файли процедура Ремпіє відкриває лише для запису, а бінарні - для читання і запису.
- Для того щоб дописати дані до наявного текстового файла, його слід відкрити процедурою Appencl.
- Файли закриваються процедурою **Ciozе.**
- Із кожним відкритим файлом зв'язаний файловий покажчик, що вказує на той компонент файла, над яким буде здійснено наступну операцію зчитування або запису. При виконанні такої операції файловий покажчик буде зсунений на наступний компонент. Під час відкриття файла файловий покажчик встановлюється на початок файла, тобто на компонент із порядковим номером 0.
- Зчитування з текстового файла виконують процедури Keacl та **KeacЛ п.** Процедура Peacl забезпечує посимвольне зчитування файла. Процедура KeacЛ **п** використовується для зчитування рядків. Запис до текстового файла здійснюється за допомогою процедур **Мгііе** та **Мпіеіп.**
- Функція **БоТ** визначає, чи файловий покажчик посилається на кінець файла.
- Зчитування з бінарних файлів здійснюється лише процедурою **Keacj,** а запис - лише процедурою Угі **ie.** До бінарного файла можна записувати значення змінних, але не можна записувати константи.
- Для прямого доступу до компонентів бінарного файла застосовують процедури **Зеек** і **Тгипсаіе,** а також функції **PilePoз** і **Pi 1 e5ire.**
- Буфером називається область пам'яті, призначена для тимчасового збереження даних під час їх передачі від джерела до приймача інформації. Застосування буферів дає можливість зменшити диспропорції між швидкостями роботи процесора та зовнішніх пристроїв.
- Розмір буфера визначається операційною системою і становить 2, 8, 16 або більше блоків. Блок є одиницею виміру обсягу даних під час обміну ними між диском та оперативною пам'яттю. Обсяг блока становить 512 байт, що є обсягом одного сектора диска.
- Відкриття нетипізованого файла для читання або запису виконується процедурами РезєТ і РеигПе, яким крім файлової змінної передається ще й додатковий аргумент - розмір файлового запису в байтах.
- Разсаі дає можливість програмісту створювати і використовувати власні буфери. Зчитування даних із фізичного файла до визначеного користувачем буфера здійснює процедура **BlockPeacj,** а запис даних з буфера до файла - процедура **BlockИгїіе.**

Контрольні запитання та завдання

1. Означити поняття логічного та фізичного файла.
2. Чим файли і масиви схожі? Чим вони відрізняються?
3. Чим відрізняються бінарні файли від текстових?
4. Чим зумовлена потреба у відкритті та закритті файлів?
5. Що таке файловий покажчик? Які стандартні процедури змінюють його значення?
6. Як здійснюється навігація по файлу?
7. Як визначається кінець файла у програмі? Як визначається кінець фізичного файла?
8. Чим відрізняється послідовний доступ до компонентів файла від прямого доступу?
9. У чому полягає відмінність між типізованими і нетипізованими файлами?
10. Як зв'язати логічний файл із фізичним?
11. Які способи відкриття файла надає мова Разсаі?
12. Які стандартні файлові змінні визначені в Разсаі?
13. Як зчитувати та записувати дані в нетипізовані файли?
14. Що таке файловий буфер і буферний пул?
15. Пояснити принцип дії механізму буферизації.

Вправи

1. Доповнити твердження.
   1.1. Текстовий файл складається із _____ змінної довжини.
   1.2. Бінарний файл - це сукупність _____.
   1.3. Кількість компонентів логічного файла не є _____.
   1.4. Для того щоб додати рядки до текстового файла, його спочатку необхідно відкрити процедурою _____.
   1.5. Процедури _____ відкривають бінарний файл для запису і читання.
2. Визначити істинні твердження.
   2.1. Компоненти файла мають один і той самий тип.
   2.2. Усі процедури запису даних до текстових файлів можна застосувати і до бінарних файлів.
   2.3. Розмір файла визначається під час оголошення файлової змінної.
   2.4. Після закриття файла зв'язок файлової змінної із фізичним файлом не переривається.
   2.5. Один фізичний файл можна зв'язати лише з однією файловою змінною.

3. Які значення до файла **probaciaT** запише наведена далі програма?

```
yar T;Ti1e oT InTедег;
    i:inTедег;
Ьеділ
  Аззідп(Т, 'rroba.ciaT');
  КемгіТе(Т);
  i :=1;
  iT eoT(T) TЬеп  мгіТе(Т,i)
            eізе Ьеділ
                   i:=i+1;  мгіТе(Т.i);
            епсі;
  iT eoT(T) Їlіеп Ьеділ
                   I:=1+1;  мгіТе(Т,I);
            епсі
            eізе Ьеділ
                   i;=i+2;  мгіТе(Т,i);
            епсі;
    CIO3Є(T);
епсі.
```

Варіанти відповідей: 1 2; 1 3; 2 3; 2 4; під час виконання програми виникне помилка.

4. Якщо файловий покажчик посилається на останній компонент файла, то виклик процедури Keacі приведе до таких результатів:
   4.1. Буде зчитаний перший компонент.
   4.2. Буде зчитаний останній компонент.
   4.3. Буде виведене повідомлення про помилку.
5. Які дії виконує процедура Беек?
   5.1. Обчислює кількість компонентів файла.
   5.2. Зсуває файловий покажчик до компонента із заданим номером.
   5.3. Повертає номер компонента, на який посилається файловий покажчик.
   5.4. Видаляє частину файла.

Задачі

1. Створити файл записів, що мають такі поля: прізвище, телефон, тривалість розмови. Вилучити із файла абонентів, тривалість розмови яких перевищує 5 год, а прізвища тих абонентів, які залишилися, відсортувати за алфавітним порядком.
2. Прямі на площині, задані рівняннями $A\hat{}x + B|y + C| = 0$ і $A_2x + B\hat{}y + C_2 = 0$, є паралельними тоді, коли $A| / By- A_2 / B_2$. Нехай **P** — файл, що містить коефіцієнти рівнянь декількох прямих. Переписати із файла **P** до файла **C** коефіцієнти рівнянь тих прямих, для яких у файлі **P** задано хоча б одну паралельну пряму.
3. Задано файл цілих чисел **P.** Використовуючи допоміжний файл **H,** переписати компоненти файла **P** до файла С так, щоб спочатку були записані всі додатні числа, а потім — всі від'ємні.

4. Створити два файли цілих чисел. Відсортувати їх вміст. Відсортовані файли злити в один впорядкований файл.
5. Створити файл записів, що мають такі поля: прізвище автора, назва твору. Видалити із файла всі записи, в яких прізвище автора починається із заданої користувачем літери.
6. Задано два текстових файли. Видалити з цих файлів рядки, що мають однакові номери, але самі не є однаковими. Результати записати до нових файлів.
7. Створити файл дійсних чисел і переписати його компоненти у зворотному порядку.
8. Задано текстовий файл *P* і рядок 5. До файла *C* записати всі рядки файла *P,* що містять рядок 5.
9. Обчислити для кожного рядка текстового файла кількість відкритих і закритих дужок і дописати обчислені значення в кінець кожного рядка. Результати записати у новий файл.
10. Створити файл записів, що мають такі поля: номер рахунку, сума внеску. Номер рахунку може вказуватися у файлі кілька разів. Створити новий файл, у якому дані про кожен рахунок будуть записані один раз, а сума внеску дорівнюватиме загальній сумі всіх внесків, зроблених на цей рахунок.
11. Задано текстовий файл, єдиний рядок якого містить 121 символ. Створити новий текстовий файл із символів першого файла за таким алгоритмом: 121 символ переписують у вигляді матриці з 11 рядками та 11 стовпцями. Потім кожен парний рядок записують у зворотному порядку; після цього у зворотному порядку записують кожен непарний стовпець.
12. Користувачеві пропонується вводити з клавіатури дані у текстовий файл. Кожен рядок файла містить назву фірми, назву товару та ціну в доларах. Перерахувати ціни у гривні за поточним курсом та дописати до рядків отримані значення. Результати записати у новий текстовий файл.
13. У кожному рядку текстового файла знайти найдовшу послідовність цифр. Значення її довжини перетворити на рядок, який записати на початку рядка вихідного файла. Результати записати у новий файл.
14. Два заданих текстових файли вважати послідовностями символів. Створити третій текстовий файл із символів, що мають однакові номери і до того ж самі є однаковими.
15. Відсортувати файл цілих чисел методом вставки, не використовуючи масиву.
16. Зашифрувати текстовий файл за допомогою заданого рядка-ключа. Кожен символ ключа додається до відповідного символу файла операцією xor. Коли символи ключа буде вичерпано, файл переглядатиметься спочатку. Що станеться, коли до зашифрованого файла повторно застосувати шифрування з тим самим ключем?
17. Вилучити коментарі з тексту записаної у файлі Pascal-програми.

Розділ 10

Динамічні структури даних

- Динамічна пам'ять, її виділення та звільнення
- Поняття покажчика і операції з покажчиками
- Робота з лінійними списками
- Поняття бінарних дерев та основні операції з ними
- Масиви у динамічній пам'яті

10.1. Динамічні змінні та динамічна пам'ять

Змінні величини, що розглядались у попередніх розділах, були *статичними.* Статичні змінні характеризуються тим, що їх значення зберігаються в ділянках оперативної пам'яті, які визначаються на етапі компіляції програми і не змінюються під час її виконання. Проте у багатьох задачах обсяг оперативної пам'яті, необхідної для збереження певних даних, неможливо визначити наперед. Для збереження таких даних використовуються змінні, які створюються і знищуються в процесі виконання програми. Такі змінні називаються *динамічними*, а пам'ять, що для них виділяється, — *динамічною пам'яттю.* Оскільки обсяг оперативної пам'яті, що використовується для збереження значення динамічної змінної, компілятору не відомий, він не позначає динамічну змінну ідентифікатором. Доступ до значення такої змінної здійснюється за її *адресою.* Отже, на етапі компіляції програми виділяється оперативна пам'ять для збереження адреси динамічної змінної, **а** пам'ять для збереження її значення виділяється під час виконання програми.

10.1.1. Розподіл оперативної пам'яті

Оперативна пам'ять комп'ютера є послідовністю байтів, або *комірок.* Розташування таких комірок є впорядкованим, і тому їх можна пронумерувати. Послідовна нумерація байтів цілими числами є зручною з погляду людини, проте процесор використовує інший спосіб доступу до комірок пам'яті - доступ за допомогою *адрес.* Адреса складається з двох шістнадцятирозрядних чисел, що називаються базисом сегмента та зсуненням. *Сегмент* — це неперервна область оперативної пам'яті обсягом 64 Кбайт (65 536 байт), що починається з комірки, номер якої є кратним 16. *Базис сегмента* - це номер 16-байтової групи, з якої починається сегмент. Таким чином, якщо базис сегмента дорівнює *x,* то цей сегмент починається з комірки, що має номер 16x *Зсунення* дорівнює відстані в байтах, на яку комірка віддалена від

початку сегмента. В адресі комірки базис сегмента та зсунення записуються, як правило, у шістнадцятковому вигляді і розділяються символом «:». Так, 000A:001A - це адреса комірки, номер якої дорівнює $(A_{16}\text{-}10_{16}) + 1A_{16} = BA_{16} = 186_{10}$ • Зазначимо, що адреси можна зіставляти не лише з окремими комірками, а і з довільними неперервними ділянками пам'яті. Адресою ділянки пам'яті вважається адреса її найпершої комірки. Принцип адресації комірок пам'яті ілюструє рис. 10.1.

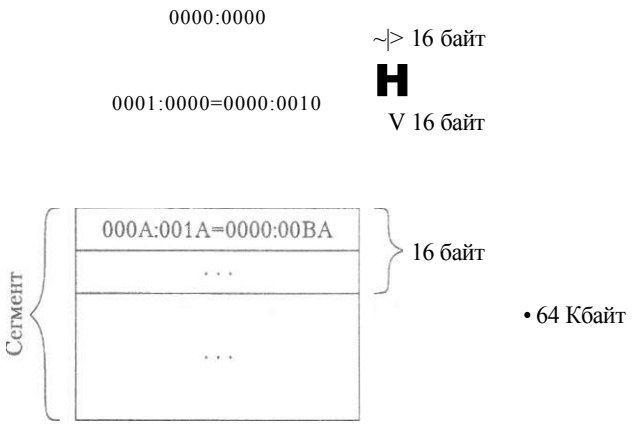


Рис. 10.1. Адресація комірок оперативної пам'яті

Середовище програмування Borland Pascal 7.0 підтримує роботу з базовою оперативною пам'яттю, обсяг якої становить 640 Кбайт. Цього обсягу достатньо для розв'язання задач, в яких використовуються структури даних досить великої вимірності. Під час роботи програми базова пам'ять комп'ютера розподіляється так.

Перед виконанням будь-якого exe-файла виділяється спеціальна область пам'яті, що називається *префіксним сегментом програми.* Дані з цієї області пам'яті, обсяг якої становить 256 байт, використовуються для керування процесом виконання програми. Зокрема, ця область містить адреси підпрограм, що здійснюють обробку переривань під час натискання клавіш Ctrl+Break, обробку критичних помилок операційної системи, завершення програми тощо. Після префіксного сегмента програми розміщується сегмент коду основної програми, обсяг якого становить 64 Кбайт. Кожному програмному модулю також виділяється сегмент коду обсягом 64 Кбайт. Крім того, сегмент коду виділяється для модуля SYSTEM, який автоматично приєднується до будь-якої програми, створеної в середовищі Borland Pascal 7.0. Типізовані константи та глобальні змінні, оголошені в розділах const; і var основної програми, розташовуються в *сегменті даних,* що має обсяг 64 Кбайт. За сегментом даних розташований *сегмент стеку,* обсяг якого становить 16 Кбайт, але може бути змінений директивою компілятора {$M}. Сегмент стеку використовується для тимчасового зберігання адреси, за якою здійснюватиметься повернення до основної програми під час виклику процедури або функції, параметрів, що передаються підпрограмі, а також для зберігання значень ого-

лошених в ній локальних змінних. Решта базової пам'яті є динамічною пам'яттю, що використовується для збереження значень динамічних змінних. Тому у загальному випадку за допомогою динамічних змінних у пам'яті можна зберігати більші обсяги даних, ніж за допомогою статичних.

Отже, динамічна пам'ять є неперервним масивом байтів. Ця область пам'яті називається *купою*, або Heap-областю (від англ. Ьеар — купа). Початкова адреса Heap-області зберігається у стандартній змінній HeapOrg, кінцева адреса - у стандартній змінній HeapEnI Адреса, якою розділяються зайнята та вільна частини купи, зберігається у стандартній змінній HeapPPr. Кожного разу після виділення динамічної пам'яті значення покажчика HeapPTr збільшується. У базовій пам'яті виділено також спеціальну область для збереження записів, що реєструють рух ділянок купи. У стандартній змінній PreePIr зберігається початкова адреса цієї області пам'яті, а кожний запис у ній містить інформацію про розташування певної динамічної змінної.

10.1.2. Поняття покажчика та його оголошення

З погляду програміста, статична змінна складається з ідентифікатора і значення. Після компіляції програми та завантаження її до оперативної пам'яті ту саму змінну можна розглядати як об'єкт, що має адресу і значення. Отже, адресу можна вважати машинним варіантом ідентифікатора змінної. Доступ до динамічних змінних здійснюється лише за їх адресами, але не за ідентифікаторами, оскільки місцезнаходження таких змінних у пам'яті стає відомим лише під час виконання програми. Для збереження адрес динамічних змінних використовуються *покажчики* - статичні змінні *посилального типу.* Значенням покажчика є адреса області пам'яті, в якій зберігається певний елемент даних. Цим елементом даних може бути значення змінної або константи, адреса іншої змінної тощо. Для збереження значення покажчика виділяється 4 байти пам'яті. У перших двох байтах записується базис сегмента, у двох інших - зсунення. Якщо область пам'яті, в якій зберігається значення змінної, складається з кількох байтів, покажчик адресує її перший байт. Зазначимо, що покажчик застосовується для посилання не лише на динамічні, але й на статичні змінні. Ця властивість покажчика робить його потужним засобом непрямого посилання.

У мові Разсаі розрізняють типізовані та нетипізовані покажчики. Покажчик, який може посилатися лише на дані певного типу, називається *типізованим,* а відповідний тип даних називається *базовим.* Для оголошення типізованого покажчика використовується символ «^», який записується перед іменем базового типу даних.

```
маг <ім'я покажчика> : ^<ім'я базового типу>;
```

Лексема ^сім'я базового типу> є ідентифікатором певного посилального типу. Проте в розділі оголошень типів даних Type для посилального типу можна оголосити і окреме ім'я, яке згодом використовуватиметься для оголошення покажчиків у розділі уаг:

```
Type сім'я посилального типу> = ^<ім'я базового типу>:
уаг сім'я покажчика> : сім'я посилального типу>;
```

Існує правило, згідно з яким будь-який ідентифікатор у мові Разсаі вперше має згадуватись у його оголошенні. Де правило не розповсюджується на ідентифікатори типів даних, що згадуються як базові типи в оголошеннях покажчиків. Інакше кажучи, можна оголосити покажчик на змінні ще не оголошеного базового типу.

Наведемо приклади оголошення типізованих покажчиків.

```
Type                          {оголошення типів                }
  yecїor    = array[1..,5] oT inTeдег:
  TiїeType  = Tiїe oT inTeдег;
  ByTePTr   = ^ByTe;          {тип покажчика на змінні типу ByTe}
  IcienTPTr = ЧcienTPec;      {тип покажчика, що посилається
                               на ще не оголошений тип запису   }
  IcienTPec = recorci         {тип запису, компонентом якого є
                               покажчик типу IcienTPTr          }
                IcienT: зTгіпд[15];
                №xT: IcienTPTr;
              enci;
  ArrayPTr  = ^УЄСТОГ;    {тип покажчика на масиви типу УЄСТОГ}
  TexTPTr   = ^TexT:      {тип покажчика на текстові файли     }
  PiIePTr   = ^Ti IeType; {тип покажчика на типізовані файли   }
yar                           {оголошення змінних              }
    pbyTe   : ByTePTr;
    prec    : IcienTPTr;
    parray  : ArrayPTr;
    pTiїe   : PiIePTr;
```

Нетипізований покажчик не зв'язується з певним типом даних і оголошується як змінна типу poinTer. Такі покажчики доцільно використовувати для посилання на дані, тип і структура яких змінюються під час виконання програми. Оголошення нетипізованих покажчиків здійснюється згідно з таким синтаксисом:

```
yar <ім'я покажчика> : poinTer;
```

10.1.3. Операції над покажчиками

Над покажчиками допустимі три операції: присвоєння, порівняння та розіменування. Розглянемо синтаксис і семантику цих операцій.

Покажчику можна присвоїти значення адреси статичної змінної або підпрограми, значення іншого покажчика, а також значення адреси, що його повертає розглянута в розділі 10.1.4 функція PTr. У такому разі для отримання адреси змінної або підпрограми використовується унарна операція визначення адреси, що позначається символом «@»:

```
«ідентифікатор покажчика> := @<ідентифікатор>;
```

Тут <ідєнтифі катор> - ім'я змінної будь-якого типу, процедури або функції.

Покажчики можна порівнювати, використовуючи при цьому операції = (рівність) та о (нерівність). Інші операції порівняння не застосовні до операндів посилального типу. Результат порівняння покажчиків, як і результат будь-якого

іншого порівняння, належить до логічного типу даних, і тому операції порівняння покажчиків можна використовувати в булевих виразах, наприклад:

```
iT Pïг1=PTг2 TTien мгiTeIпC 'роіпїегз аге едиаі');
```

Зауважимо, що порівнювати значення покажчиків, а також присвоювати значення одного покажчика іншому можна лише в тих випадках, коли:

- принаймні один із покажчиків є нетипізованим;
- обидва покажчики посилаються на один і той самий базовий тип даних.

Щоб отримати значення, на яке посилається покажчик, потрібно виконати операцію *розіменування покажчика.* Ця операція позначається символом «^{A}», що записується після імені покажчика. Її можна записувати як у правій, так і в лівій частині оператора присвоєння:

```
«ідентифікатор змінноі> := «ідентифікатор покажчика^;
«ідентифікатор покажчика^ := «вираз>:
```

Операцію розіменування можна застосовувати лише до покажчиків, яким надано значення певної адреси динамічної пам'яті. Розглянемо дві змінні: змінну yагiaЬIe типу iпТедег і покажчик ріг на дані типу iпТедег. Нехай покажчику pTг надано адресу змінної yагiaЬIe, тобто виконано операцію pTг:=@yапаЬ1е. Тоді значення виразів yагiaЬIe і pTг^{A} завжди будуть однаковими. Якщо ж присвоєння pTг:=@yагiaЬIe замінити операцією pTг^{л}:=yагiaЬ1е, то змінні pTг^{A} та yагiaЬIe будуть різними, хоча, можливо, і матимуть протягом певного часу однакові значення. У прикладі 10.1 наведені й інші варіанти використання операцій із покажчиками.

Зазначимо, що результат розіменування нетипізованого покажчика має невизначений тип, і тому вираз р1г^{A}, де ріг — нетипізований покажчик, не може входити до складу інших виразів.

Нарешті, спробуємо встановити, яке значення має ще не ініціалізований покажчик. У більшості випадків компілятор надає йому спеціальне значення, що позначається зарезервованим словом пі 1. Якщо значення певного покажчика становить пП, то вважається, що покажчик не посилається на жоден елемент даних. Тому значення пі 1 називають ще *порожньою адресою.* Результат розіменування покажчика, значення якого дорівнює пі 1, є непередбачуваним.

Приклад 10.1

Наведемо програму, що призначена для демонстрації принципів використання операцій над покажчиками. Зазначимо, що покажчики не можна використовувати як аргументи підпрограм. Із цього, зокрема, випливає висновок про неможливість надрукувати значення покажчика за допомогою стандартних процедур мгiTe і игiTeїп. Спроба зробити це призведе до появи помилки **Error 64: CannoT** Яеасі or **\AhiTe yariaЬIes oT THis Type** (Не можна вводити або виводити змінні даного типу.). Тому значення покажчиків, використані в програмі ex10_1, можна переглядати лише під час налагодження програми у вікні \A/aTcli. Зв'язок між покажчиком і динамічною змінною, на яку він посилається, проілюстрований на рис. 10.2. Результати роботи програми ex10_1 зображено на рис. 10.3.

```
рrодгат ex10_1;
11565 crT;
Type
  уєстог = array [1..5] oT inTедег;
  TypeTiIe = Tiїe oT inTедег;
  IcienTPTr = "IcienTKec;
  IcienTKec = recorci
       IcienT; зTrinд[15];
       №xї; IcienTPTr;
  enci;
yar pbyTe : "ByTe;
  pinTI, pinT2 : "inTедег;
  parray : "уесTог;
  pTiїe : "TypeTПe;
  инTypeci : poinTer;
  prec : IcienTPTr;
  x : ЬyTe:
  y : inTедег;
  arr : уєстог;
  T : Tiїe oT inTедег;
  1izT : IcienTKec;
Ьеdiп
  CIrзcr
  y := 1
  pinTI := @y;
  y : = 2
  pinT2 := @y;        {покажчики pinTI та pinT2
                       посилаються на ту саму змінну - y    }
  мгiїе('pinTI anci pinT2 are TЬe зате: [1]);
  мгiTe1п('(pinT1=pinT2)=',pinT1=pinT2);  {порі вняння
                                          покажчикі в}
  мгiTe1п('pinT1"='.pinTI",' pinT2"=',pinT2"," y='.y):
  pbyTe := @x;
  инTypeci := pbyTe;  {використання нетипізованого          }
  pinTI := инTypeci;  {покажчика, pinTI та pbyTe             }
  x := 4;             {посилаються на одну й ту саму змінну}
  и/giTe1п( 'pbyTe"=' .pbyTe"," pinT1"='.pinTГ,[1] x=',x);
  {використання покажчиків для посилання
   на елементи масивів та записів}
  parray := @arr;
  parray"[1] : = 1:
  prec := @1izT;
  prec".Icienї := 'PoinTerз !!!';
  мгiTe1п('arr[1]=',arr[1].' parray"[1]=',parray"[1]);
  мгiTe( 'IizT. IcienT='.IizT. IcienT);
  мгiTeC' prec". IcienT=' .prec".IcienT);
  {використання покажчика на файлову змінну}
  аззідп(T,'пемTПe.Tхї');
  pTi1e:=@T;
  гем~iTe(T);
  мг1Te(pTПe",y);    {до файлу 'пемTПe.TxT' буде записано}
  c1oзe(pTi!e");     {значення змінної y                  }
  геасії п;
enci.
```

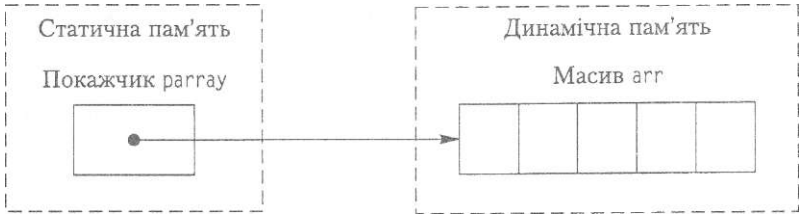


Рис. 10.2. Покажчик у статичній пам'яті та змінна в динамічній пам'яті

```
• C:\BP\BIN\EX 10_1.EXE                                    B B I
pinИ anri pin12 ace Ibe зате: (pin(:1='pin1:2) = TPIIE
pinI:i"=2 pinï2"=2 y=2
pbyf£e"=i» piniï"^
na5[1]=1 parray"[1]=1
1!5!.Iйen(:=Poinler5 ••Г prec".Iйen(:=PoinI:er5 •?»
```

Рис. 10.3. Результати роботи програми ex10_1.
Операції над покажчиками

10.1.4. Виділення та звільнення динамічної пам'яті

У мові Разсаі використовуються три методи роботи з пам'яттю, яка динамічно розподіляється:

4 за допомогою процедур **i Olspose;**

• за допомогою процедур **CeTMaш i PreeMer;**

4- за допомогою процедур Маік **i Peiease.**

Процедура Иєм виділяє область динамічної пам'яті з урахуванням типу покажчика і присвоює адресу пам'яті, що виділена, цьому покажчикові. Отже, процедура створює динамічну змінну того типу, що є базовим для покажчика. Якщо покажчик посилається на тип даних, для якого потрібно пам'яті більше, ніж доступно для розподілу в Неар-області, компілятор генерує помилку Ептог 203: Неар оуєгПоуу **ептог** (Переповнення купи). Синтаксис виклику процедури Кєм є таким:

№м(<ідентифікатор покажчика>);

Процедуру №м можна викликати як функцію, що повертає значення покажчика на тип даних, який передається процедурі як параметр. Наприклад., якщо означено посилальний тип іпсі1са1;ог="іпіедег і покажчик ррг: і псіі саРог, то наступні два оператори еквівалентні:

№м(рІг);

рїг := Мєм(1пclicaiог):

Використання покажчиків, яким не було надано певного значення за допомогою процедури Мєм або операції отримання адреси може призвести до непередбачуваних наслідків.

Процедура йізрозе звільняє область динамічної пам'яті, на яку посилається її параметр-покажчик, після чого ця область стає доступною для розподілення між іншими динамічними змінними. Отже, процедура Оізрозе видаляє динамічну змінну, що адресується її параметром-покажчиком. Синтаксис виклику процедури От зрозе такий:

```
01зро5є(<ідентифікатор покажчика>);
```

Процедура Оізроге не змінює значення покажчика, а лише повертає до купи пам'ять, що раніше була з ним зв'язана. Застосування процедури до порожнього покажчика призведе до виникнення помилки **Error 204: Invalid pointer operation** (Неприпустима операція з покажчиком.).

Для роботи з нетипізованими покажчиками можна використовувати процедури йеіМеш і PreeMet. Процедура СеїМет виділяє область динамічної пам'яті заданого обсягу. Початкова адреса області пам'яті, яка буде виділена, запам'ятовується у покажчику, що є параметром процедури. Виклик процедури СеіМет має такий синтаксис:

```
беїМеш(<ідентифікатор покажчика>,<обсяг пам'яті>);
```

Тут «ідентифікатор покажчика> — змінна типу роіпіег; «обсяг пам'яті> — змінна або вираз типу иогі Обсяг пам'яті задається в байтах. Обсяг пам'яті, що виділяється для однієї динамічної змінної, не може перевищувати 65 521 байт. Крім того, не можна виділяти ділянок пам'яті, довжина яких перевищує довжину найбільшої вільної ділянки в купі, що її повертає функція MaxAyaiI. Тому для забезпечення коректності роботи процедури ЕеіМепі бажано перед її викликом перевіряти наявність достатньо великої вільної ділянки пам'яті

```
1Г МахДуаі 1 >9999 Уіеп
  беїМеяКрїг, 10000)
еізе
  мгПеСІІо зиТТісіепІ тетогу');
```

Зазначимо, що загальний обсяг усіх вільних ділянок динамічної пам'яті обчислює функція МешАуаі 1.

Процедура PreeMet звільняє пам'ять, адресовану покажчиком, який є параметром процедури. Обсяг пам'яті, що звільняється, зазначається як другий параметр процедури PreeMet. Наведемо синтаксис виклику процедури:

```
PreeMet(<ідентифікатор покажника>,«обсяг пам'яті>):
```

Слід пам'ятати, що розподіл динамічної пам'яті за допомогою процедури СеіМет потребує її звільнення за допомогою процедури PreeMet. Під час застосування цих процедур до одного й того самого покажчика значення параметрів, що задають обсяг динамічної пам'яті, мають збігатися.

У результаті багатьох викликів процедур і Оізрозе, а також беІМет і PreeMet динамічна пам'ять фрагментується. В ній з'вляються несуміжні вільні ділянки. Щоб мати можливість звільняти динамічну пам'ять, використовуються процедури Маік і РєІеазе.

Синтаксис виклику цих процедур такий:

```
Магк(<1 дентифі катор покажчика>);
Реїеазе(<ідентифікатор покажчика>);
```

Процедура Магк запам'ятовує поточне значення покажчика HeapPїг у покажчику, що є параметром процедури. Нагадаємо, що покажчик HeapPіг містить адресу початку вільної динамічної пам'яті. Процедура Реїеазе звільняє динамічну пам'ять, починаючи від комірки, що адресується параметром процедури, до кінця динамічної пам'яті. Один виклик процедури Реї еазе знищує список усіх вільних фрагментів у динамічній пам'яті, створених викликами процедури Оізрозе, а також усі динамічні змінні, створені після виклику процедури Магк. Використовувати два різні механізми звільнення динамічної пам'яті (за допомогою процедур Оі зрозе і Реї еазе) в межах однієї програми небажано, оскільки доведеться стежити за тим, щоб процедура Оізрозе не звільняла змінні, що вже були звільнені процедурою Реї еазе.

Приклад 10.2_

Програма ex10_2 демонструє механізм роботи процедур і Оізрозе, а також Магк і Реї еазе. Стан динамічної пам'яті під час роботи програми показано на рис. 10.4.

```
ргодгат ex10_2;
уаг
  pir,p1,p2,p3.p4:"iпТедег;
Ьеdіп
  №м(p1);     {виділити пам'ять для цілого числа          }
  Магк(рїг);  {запам'ятати адресу початку вільної області}
  №м(p2);     {виділити пам'ять ще для трьох цілих чисел }
  №м(p3);
  №м(p4);
  Оізрозе(p3);   {звільнити пам'ять від p3"
  Pelеазе(pTr); {звільнити пам'ять від p2",p3", p4"
  {0ізрозе(p4);   помилка: змінну p4" вже видалено!!!
  геасії п;
епсі.
```

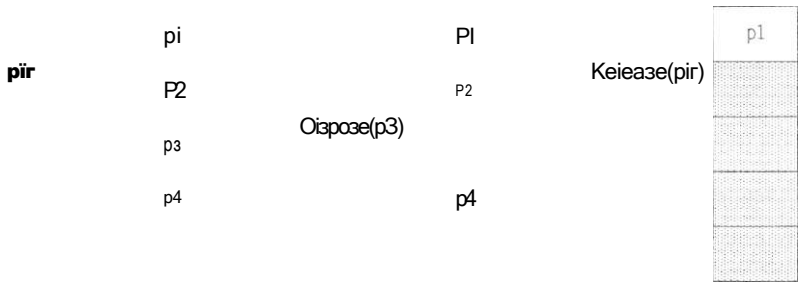


Рис. 10.4. Стан динамічної пам'яті після розподілу (а) та після звільнення процедурами Різрозе (б) і Веіеазе (в)

10,1.5, Стандартні функції для робота з адресами

Мова Разсаі не дає можливості записувати адреси як константи, а також виконувати арифметичні операції безпосередньо над адресами. Проте ці дії можна виконати за допомогою стандартних функцій, які будуть розглянуті нижче.

- Функція Acicir використовується для отримання адреси змінної або підпрограми і може застосовуватися замість операції визначення адреси Формат виклику функції Acicir є таким:

  «ідентифікатор покажчика> := Acicir(<і м * я змінної або п1дпрограми>);

4 Функція Сзед повертає значення типу иогсі, що зберігається в регістрі С5 процесора. Коли програма починає роботу, в регістр С5 завантажується базис сегмента коду програми. Функція не має параметрів.

4 Функція Озед повертає значення типу иогсі, що зберігається в регістрі 05 процесора. Коли програма починає роботу, в регістр 05 завантажується базис сегмента даних програми. Функція не має параметрів.

4 Функція ОТз повертає значення типу могсі. Це значення дорівнює зміщенню адреси змінної, яка зберігається в сегменті даних, або зміщенню адреси підпрограми, що записана у сегменті коду. Синтаксис виклику функції такий:

«ідентифікатор змінної типу тогсі> := 0Тз(«1м'я змінної або підпрограми>);

4 Функція Ріг перетворює пару цілих чисел — значення базису сегмента та значення зсунення - на адресу, тобто на значення типу роіпіег. Виклик функції має такий синтаксис:

«і дентифі катор покажчика>: =Ріг(«5едтепі:>. <0ТТзе£>);

Тут «ідентифікатор покажчика> - ідентифікатор змінної типу poinier; <5едтеп1;> — базис сегмента; <0ТТзе£> - зсунення елемента даних у межах сегмента.

10.2. Спискові структури даних

Використання покажчиків є ефективним способом побудови динамічних структур даних. У розділах 7-9 вже розглядалися структури даних, такі як масиви, множини, записи та файли. Ці структури є статичними, тому що їхні розміри визначаються на етапі компіляції і залишаються незмінними протягом усього часу виконання програми. Розміри динамічних структур даних визначаються і можуть змінюватися під час виконання. Основними різновидами динамічних структур є лінійні списки, дерева (нелінійні списки) і динамічні масиви,

10.2.1. Визначення лінійного списку та його різновидів

Як приклад розглянемо таку задачу. Здійснюється реєстрація автомобілів, які прибувають на автостоянку та залишають її. Потрібно зберігати і обробляти множину номерів автомобілів. Для відображення цієї множини в пам'яті комп'ютера

необхідно обрати певну структуру даних. Вибір масиву буде невдалим з декількох причин. По-перше, умова задачі не обмежує кількість автомобілів, а розмір масиву є обмеженим. По-друге, в разі прибуття кожного нового автомобіля на автостоянку або його від'їзду потрібно буде вставити або видалити елемент масиву. Така операція є досить трудомісткою, оскільки потребує послідовного зсуву в пам'яті значної кількості елементів. Вирішити ці проблеми можна шляхом використання динамічної структури даних, яка називається зв'язним лінійним списком.

Зв'язний лінійний список - це сукупність однотипних компонентів, які послідовно зв'язані між собою за допомогою покажчиків. Кожен компонент списку, крім останнього, містить покажчик на наступний (або на наступний і попередній) компонент. Доступ до першого компонента здійснюється за допомогою покажчика на нього, а доступ до кожного наступного компонента — з використанням покажчика, який зберігається у попередньому компоненті. Перший компонент списку називається його *вершиною*, або *головою.*

Над зв'язними лінійними списками виконуються такі дії:

- додавання нового компонента на початок списку;
- додавання нового компонента в кінець списку;
- вставка нового компонента між двома наявними компонентами списку;

4 видалення компонента зі списку.

Зв'язні лінійні списки поділяють на такі різновиди:

- однозв'язний лінійний список;
- двозв'язний лінійний список;
- однозв'язний циклічний список;
- двозв'язний циклічний список;
- стек;
- черга.

Однозв'язний лінійний список — це список, в якому попередній компонент посилається на наступний.

Двозв'язний лінійний список — це список, в якому попередній компонент посилається на наступний, а наступний — на попередній.

Однозв'язний циклічний список - це однозв'язний лінійний список, в якому останній компонент посилається на перший.

Двозв'язний циклічний список - це двозв'язний лінійний список, в якому останній компонент посилається на перший, а перший компонент - на останній.

Стек — це однозв'язний лінійний список, в якому компоненти додаються та видаляються лише з його вершини, тобто з початку списку.

Черга — це однозв'язний лінійний список, в якому компоненти додаються в кінець списку, а видаляються з вершини, тобто з початку списку.

Графічне зображення однозв'язного лінійного списку ви бачите на рис. 10.5.

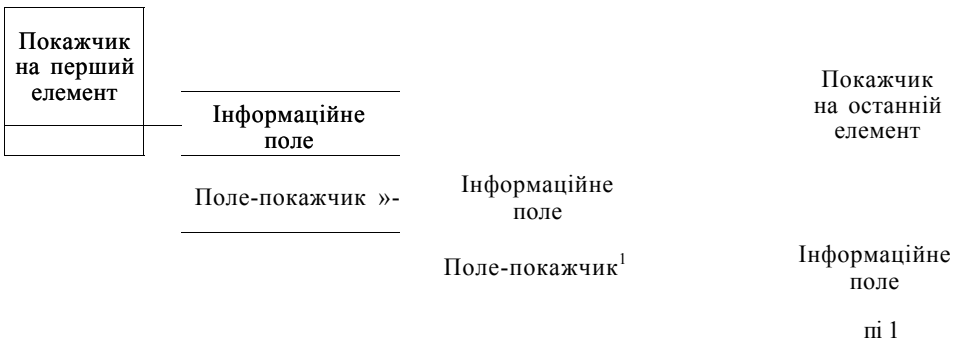


Рис. 10.5. Однозв'язний лінійній список

Опишемо тип компонентів зв'язного лінійного списку. Почнемо з однозв'язного лінійного списку. Кожний його компонент складається з кількох інформаційних полів та покажчика на наступний компонент. Отже, компонент зв'язного лінійного списку є записом. Інформаційні поля компонента списку можуть бути змінними будь-яких типів, а покажчик повинен бути покажчиком на запис того типу, якому належать компоненти списку. Покажчик в останньому компоненті лінійного списку має значення пі 1 — так позначається кінець списку.

Тип покажчика на компонент однозв'язного лінійного списку має бути оголошений перед оголошенням типу компонента списку. Таке виключення з правил зроблено спеціально для типів компонентів динамічних структур.

Наведемо приклад оголошення типу компонента однозв'язного лінійного списку. У найпростішому випадку компонент містить одне інформаційне поле, у якому може зберігатися, наприклад, реєстраційний номер автомобіля. Для роботи з таким списком потрібні покажчики на перший і поточний компоненти. Оголосимо ці покажчики після оголошення типу компонента списку.

```
Type
  piг="Гiет;              {ТИП покажчика на компонент списку}
  Їет=гесогсі             {тип компонента                   }
        сіаіа   зігіпд;   {інформаційне поле                }
        пехї    piг;      {покажчик на наступний компонент }
      епсі
yar                       {покажчики на перший та           }
   iieacісигг : piг;      {поточний компоненти списку       }
```

10.2.2. Робота зі стеком

Стек - це один із різновидів однозв'язного лінійного списку, доступ до елементів якого можливий лише через його початок, що називається *вершиною стеку.* Стек працює за принципом «останнім прийшов — першим вийшов», що позначається абревіатурою ЬІРО (від англ. Ьаїі Іп Рігзі Оиі), і має такі властивості:

- елементи додаються у вершину (голову) стеку;
- елементи видаляються з вершини (голови) стеку;

- покажчик в останньому елементi стеку дорiвнює пi I;

4- неможливо вилучити елемент із середини стеку, не вилучивши всi елементи, що йдуть попереду.

У програмуванні стеки мають широке застосування. Наприклад, під час виклику підпрограми адреса повернення до неї зберігається у стеку. Стек використовується компілятором під час обчислення виразів, до нього записуються значення локальних змінних тощо.

Для роботи зі стеком достатньо мати покажчик пеао на його вершину та допоміжний покажчик сиггепі на елемент стеку. Наведемо алгоритми основних операцій зі стеком — вставки та видалення його елемента.

Алгоритм вставки елемента до стеку

1. Виділити пам'ять для нового елемента стеку: пем(сиггепі) (рис. 10.6, *а).*
2. Ввести дані до нового елемента: геасіїп(сиггепI^{A}.ciaia) (рис. 10.6, *б).*
3. Зв'язати допоміжний елемент із вершиною: сиггепі".пехї:-Неес (рис. 10.6, *в).*
4. Встановити вершину стеку на новостворений елемент: Ьеасі: =сиггепI (рис. 10.6, *г).*

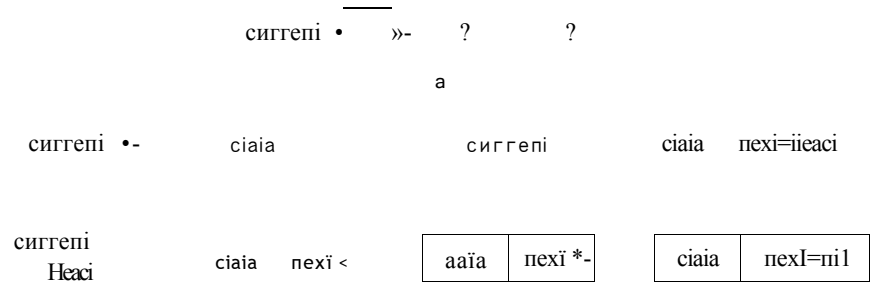


РИС. 10.6. Вставка елемента до стеку: виділення пам'яті (а); введення даних (б); **зв'язування з вершиною** (в); переміщення вершини (г)

Зауважимо, що значенням покажчика Meacі на вершину порожнього стеку є пі I. Тому для створення стеку слід виконати оператор Ьеасі:=пі1 та повторити щойно наведений алгоритм потрібну кількість разів.

Алгоритм видалення елемента із непорожнього стеку

1. Створити копію покажчика на вершину стеку: сиггепі :=пеасі; (рис. 10.7, *а).*
2. Перемістити покажчик на вершину стеку на наступний елемент: Неасі:=сиггепі".пехI (рис. 10.7, *б).*
3. Звільнити пам'ять із-під колишньої вершини стеку: 0i5po5e(сиггепI) (рис, 10.7, в).

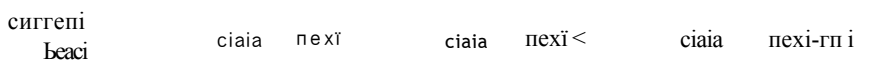


Рис. 10.7. **Видалення** елемента із непорожнього стеку: **створення** копії покажчика на вершину стеку (а)

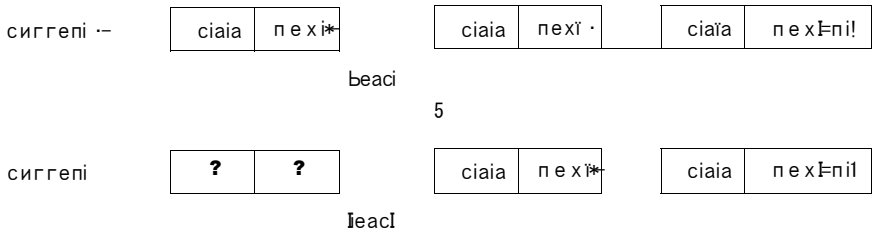


Рис. 10.7 (*продовження):* переміщення вершини (б); звільнення пам'яті *(в)*

Зрозуміло, що для очищення всього стеку слід повторювати кроки 1-3 доти, доки покажчик iieacі не дорівнюватиме пії.

Приклад 10.3_

У програмі ex10_3 використовується меню, за допомогою якого можна вибрати команду додавання до стеку групи елементів, або команду відображення та видалення стеку. Процедура ризії викликається для того, щоб додати до стеку елемент, значення якого вводить користувач. Для збереження введеного значення використовується змінна зіг. Вершина стеку зберігається у змінній HeacIГ, а покажчик сиггепі використовується як допоміжний. Процедура pop викликається для видалення елемента зі стеку і записує його значення до параметра-змінної yai ие. Нижче, на рис. 10.8, зображено результати роботи програми ex10_3.

```
ргодгат е х10_3;    {створення, виведення та очищення стеку}
изез с г і;
Iуре
   рІг=^IIе т;            {тип покажчика на елемент стеку
   IIет=гесогсі           {тип елемента стеку
        ciaia ізігіпд:    {інформаційне поле елемента
        пехї:ріг;         {покажчик на наступний елемент
       епсі;
уаг HeacI                {вершина стеку
    сиггепі:ріг;         {допоміжний покажчик
    зіг;зігіпд;          {значення, що додається до стеку
    і Ипіедег;           {параметр циклу
    п:іпIедег:           {кількість елементів, що додаються
    кеу:сИаг;            {номер пункту меню програми
(==========д о д а ванн я елемента до с т е к у==========
ргосесіиге ри зіі(уа1ие:з1гіпд);
                         {параметр - значення, що додається
Ьеділ
  пем(сиггепі);          {виділити пам'ять для елемента
  сиггепї^. (Ма:=уа1ие; {і ні ці алі зувати інформаційне поле
  сиггепї^.пехї=iieacI; {зв'язати елементи стеку
  IieacI=сиггепI;        {встановити нову вершину стеку
епсі;
```

```
{====== видалити елемент зi стеку та зберегти його =====}
procedure pop(var a: Integer);
          {параметр - змiнна, значення якої буде виведене}
begin
  current := head;          {зберегти адресу вершини стеку }
  a := head^.info;          {зберегти значення, що виводиться}
  head := current^.next;    {перемiстити вершину на другий
                             елемент                        }
  dispose(current);         {ЗВIЛЬНИТИ пам'ять             }
end;
{=============основна програма=================}
begin
  init;
  menu := true;
  writeln('СТЕК');
  head := nil;                    {стек порожнiй          }
  repeat                          {вивести меню           }
    writeln('1. додати елемент до стеку');
    writeln('2. вивести та очистити стек');
    writeln('3. вихiд');
    writeln('Введiть номер вибраного пункту меню');
    readln(key);                  {вибрати пункт меню     }
    case key of
      '1': begin                  {додати групу елементiв}
             writeln('введiть кiлькiсть елементiв');
             readln(n);
             for i := 1 to n do
             begin
               writeln('введiть елемент ',i);
               readln(a);
               push(a);           {додати елемент до стеку }
             end;
           end;
      '2': begin                  {вивести та очистити стек}
             while head <> nil do
             begin
               pop(a);            {видалити елемент зi стеку}
               WRITE(A,' ');{вивести значення елемента}
             end;
             writeln;
             writeln('стек порожнiй');
           end;
    end
  until key='3';                  {ПОКИ не обрано вихiд iз програми}
end.
```

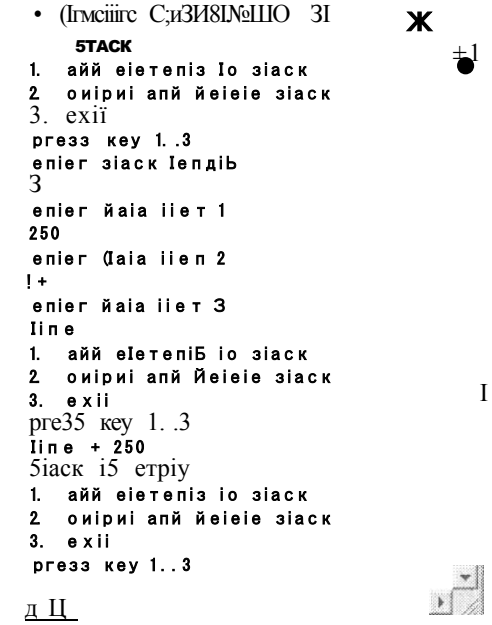


Рис. 10.8. Результати роботи програми ex10_3. Обробка стеку

10.2.3. Робота з чергою

Черга, як і стек, — це один із різновидів однозв'язного лінійного списку. Вона працює за принципом «першим прийшов — першим вийшов», що позначається абревіатурою PIPO (від англ. Рігзї Іп Рігзі Оиї), і характеризується такими властивостями:

4 елементи додаються в кінець черги;

Ф елементи зчитуються та видаляються з початку (вершини) черги;

• покажчик в останньому елементі черги дорівнює пі 1;

неможливо отримати елемент із середини черги, не вилучивши всі елементи, що йдуть попереду.

Наведемо приклади застосування черг в обчислювальній техніці. У мережній операційній системі процесор сервера обслуговує в певний момент часу тільки одного користувача. Запити інших користувачів записуються до черги. Під час обслуговування користувачів кожен запит просувається до початку черги. Перший в черзі запит підлягає «першочерговому» обслуговуванню. У комп'ютерній мережі за чергою обслуговуються інформаційні пакети. Черги застосовуються також для буферизації потоків даних, що виводяться на друк, якщо в комп'ютерній мережі використовується один принтер.

Для роботи з чергою потрібні: покажчик HeacI на початок черги, покажчик Iазі на кінець черги та допоміжний покажчик сиггепі;. Зауваживши, що елементи з черги видаляються за тим самим алгоритмом, що і зі стеку, наведемо алгоритм вставки до черги нового елемента.

Алгоритм вставки елемента до черги

1. Виділити пам'ять для нового елемента черги: пеи(сиггепї) (рис. 10.9, *а).*
2. Ввести дані до нового елемента: геасЛп(сиггепї".ciaia) (рис. 10.9, *б).*
3. Вважати новий елемент останнім у черзі: сиггепі".пехї:=пil (рис. 10.9, *в).*
4. Якщо черга порожня, то ініціалізувати її вершину: HeacI:=сиггепі.
5. Якщо черга не порожня, то зв'язати останній елемент черги із новоутвореним: 1азі".пехї:=сиггепі.
6. Вважати новий елемент черги останнім: 1азї:=сиггепї (рис. 10.9, *г).*

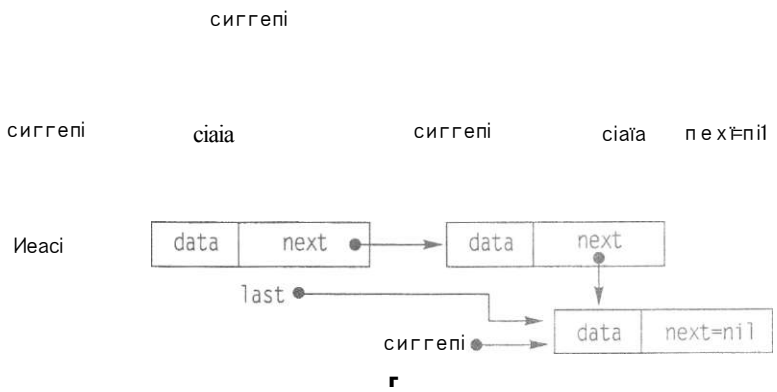


РИС. 10.9. Вставка елемента до черги: виділення пам'яті (а); введення даних (б); **встановлення ознаки кінця черги (в); переміщення** кінця **(г)**

Приклад 10.4

Програма ex10_4 має структуру, подібну до структури програми ex10_3. Процедура acici викликається для додавання до черги елемента, значення якого вводить користувач. Для збереження введеного значення використовується змінна зїг. Вершина черги зберігається у змінній iieacГ, кінець — у змінній Iазї", а покажчик сиггепі є допоміжним. Процедура деїеі викликається для видалення елемента із черги і записує значення елемента, що видаляється, до параметра-змінної уаіие. Результати роботи програми ex10_4 зображено на рис. 10.10.

```
ргодгат ex10_4; {створення, виведення та очищення черги}
изез сгї;
їуре
   рїг=^Їїет:              {ТИП покажчика на елемент черги }
   Їїет=гесогсі            {тип елемента черги            }
        cIaїa:зіг1пд:  {інформаційне поле елемента    }
        пехї:рїг;     {покажчик на наступний елемент }
      епсі:
```

```
уаг Неасі                {покажчик на початок черги
    сиггепі:ріг;         {допоміжний покажчик
    1азі:ріг;            {покажчик на кінець черги
    5Іг:ЗІгіпд;          {значення, що додається до черги
    і:іпіедег;           {параметр циклу
    п:іпІедег;           {кількість елементів черги
    кеу:сІіаг;           {номер пункту меню програми
{                додати елемент до черги ================
ргосесіиге асісі(уа1ие5Ігіпд);
                         {параметр - значення, що додається
Ьеділ
  пем(сиггепі);          {виділити пам'ять для елемента
  сиггепі^.сіаІа:=уа1ие; {і ні ці алі зувати інформаційне поле
  сиггепі^.пехї:=піІ;    {встановити ознаку кінця черги
  іі 1азІ=пі1            {якщо створюється перший елемент,
      іііеп              {і ні ціалі зувати
        Иеасі:=сиггепІ   {покажчик на початок черги
      еізе               {якщо новий елемент не є першим,
       зв'язати колишній кінець черги із новим елементом
        1азі^.пехї:=сиггепІ;
  1азі:=сиггепі;     {новий елемент вважати останнім
епсі;
{====== видалити елемент із черги, зберігши його =======
ргосесіиге деІсіеКуаг уа1ие:зігіпд);
Ьеділ
  сиггепі:=НеасІ;    {зберегти адресу вершини черги
  уа1ие:=ІеасГ.сіаіа; {зберегти значення, що виводиться
  Ьеасі:=сиггепІ^.пехї; {перемістити вершину черги
                         на другий елемент
  іІ Иеасі=пі1           {якщо черга стала порожньою,
      іЬеп 1азІ:=пі1     {порожнім має стати і кінець черги
  сііЗрозе(сиггепі);     {звільнити пам'ять
епсі;
{    —    —         основна програма
Ьеділ
  сігзсг;
  мгііеІпС    ОІІЕІІЕ');
  Ьеасі:=пП ;                          {черга порожня
  1азІ:=пі1;
  гереаі
    мгііеІпСІ. сгеаіе диеие');
    мгіІе1п('2. оиіриі апсі сіеіеіе диеие'):
    мгіїеІпС3. ехіі');
    мгіІе1п(ргезз кеу 1..3');
    кеу :=геасікеу:
    сазе кеу оі
      '1': Ьеділ                  {додати елемент до черги}
             мгііе!п(епіег ^иеие Іепдіь');
             геаспп(п);
             іог і :=1 іо п сіо
             Ьеділ
```

```
            мгiie1п('enier ciaia iiet ¥i);
            reacï п(5їг);
            acM(БЇг);
          encs;
        encs:
    '2': Ьедіп                {вивести та видалити елемент}
          мМIe Heaciопії cio
          Ьедіп
            деiciei (зіг);
            мгiie(зіг.' ');
          encs;
          мгiieіп;
          мгiie!п(диеие із ешріу');
        encs;
      encs
      иniii key="3';
encs.
```

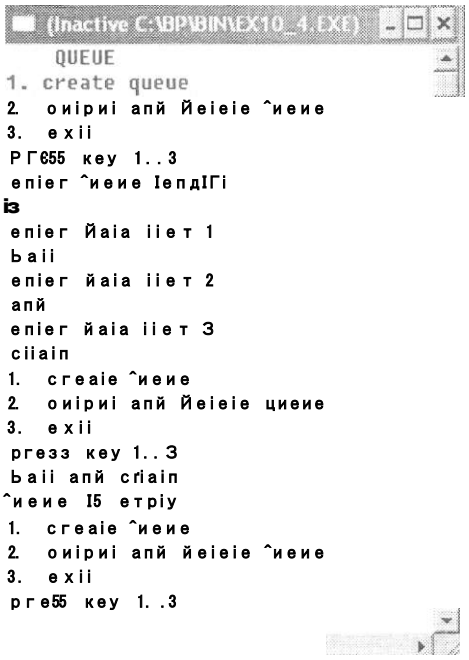


РИС. 10.10. Результати роботи програми ex10_4. Робота з чергою

10.2.4. Робота з лінійним списком

Розглянуті у попередніх двох розділах стек і черга є лінійними списками, множина допустимих операцій над якими обмежена операціями над першим або останнім елементом. У даному розділі обговорюватимуться списки, над якими припустимі

довільні дії. Найбільш ефективно у спискових структурах реалізуються операції вставки та видалення елементів, оскільки вони, на відміну від операцій видалення та вставки елементів масиву, не потребують зсуву групи елементів. Наведемо всі можливі варіанти застосування цих двох операцій:

- створення списку, тобто внесення першого елемента до списку;
- додавання елемента в кінець списку;
- додавання елемента на початок списку;
- вставка елемента в середину списку;
- видалення елемента з початку списку;
- видалення елемента з кінця списку;
- видалення елемента з середини списку.

У загальному випадку для роботи з однозв'язним лінійним списком потрібні такі покажчики: покажчик іїеасі на початок списку; покажчик сиггепі на **поточний** елемент списку; покажчик ргеуіоиз на елемент, розташований перед поточним; покажчик пемріг на елемент, що додається до списку, та покажчик Іазі на кінець списку. Зауважимо, що у розв'язаннях конкретних задач можуть використовуватися не всі такі покажчики. Зокрема, у наведеному нижче прикладі 10.5 зайвим виявиться покажчик Іазі.

Отже, розглянемо алгоритми основних операцій над однозв'язним лінійним списком. Нагадаємо, що кожний елемент однозв'язного списку можна реалізувати записом Іїет, оголошення якого наведене в розділі 10.2.1. Додавання елемента в кінець списку виконується за алгоритмом додавання елемента до черги, а додавання елемента на початок списку — за алгоритмом додавання елемента до стеку. Ці операції вже розглядалися в двох попередніх розділах.

Так само вже розглядалася і операція видалення елемента з початку списку, що здійснюється за алгоритмом видалення елемента зі стеку або з черги. Запишемо алгоритми решти операцій, припускаючи, що при додаванні елемента для нього вже була створена динамічна змінна пемріг" та було введене значення у його поле сіаіа.

Алгоритм створення одноелементного списку

1. Ініціалізувати початок списку: іїеасі:=пемрїг.
2. Ініціалізувати кінець списку: 1 азі:: =пемр"Ьг.
3. Записати ознаку того, що перший елемент є останнім: ІїеасГ.пехІ:=пі1.

Алгоритм вставки елемента всередину списку

Вважаємо, що новий елемент має бути вставлений між елементами ргеуіои5" і сиггепі" (рис. 10.11).

1. Новий елемент вважати наступним для ргеуіоиз": ргеуіоиз".пехІ:= пемрїг.
2. Елемент сиггепі" вважати наступним для нового елемента: пемріг". пехї: = сиггепі.

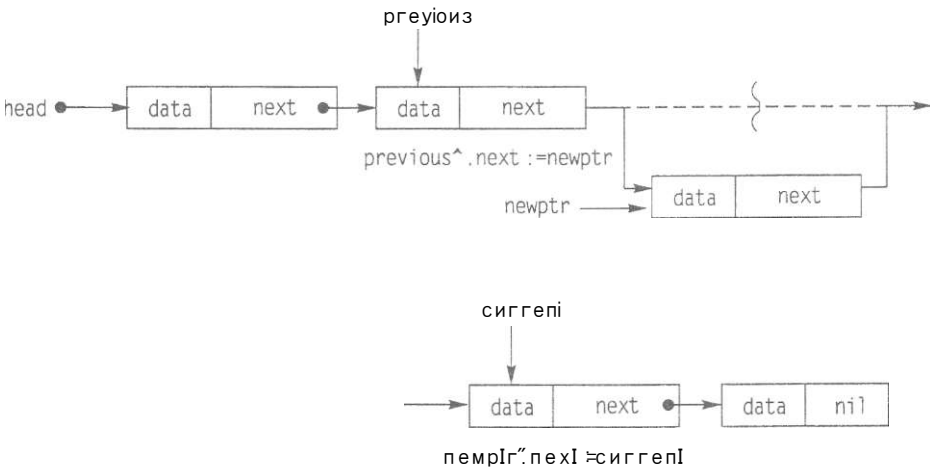


РИС. 10.11. Вставка елемента всередину списку

Алгоритм видалення елемента зсередини списку

Вважаємо, що видаляється елемент сиггепї", розташований безпосередньо за елементом pгeyioиз" (рис. 10.12).

1. Вважати, що за елементом pгeyioиз" буде розташований той елемент, що раніше знаходився за елементом сиггепї": pгeyioиз".пехї:=сиггепї".пехї;
2. **звільнити** пам'ять із-під елемента сиггепї": Oізрозе(сиггепі);

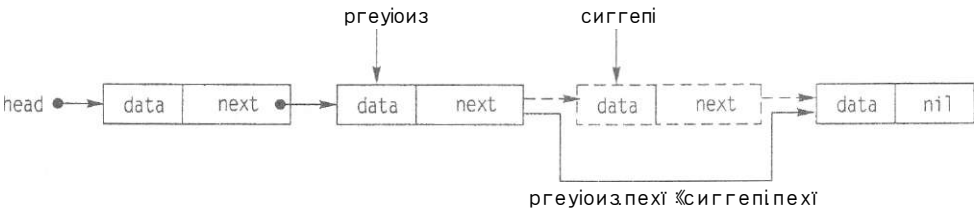


РИС. 10.12. Видалення елемента зсередини списку

Алгоритм видалення елемента з кінця списку

Вважаємо, що на передостанній елемент посилається покажчик pгeyioиз.

1. Записати до передостаннього елемента ознаку кінця списку: pгeyioиз".пехї: = пії.
2. **звільнити** пам'ять із-під колишнього останнього елемента: Oізрозе (Іазі).
3. Вважати останнім колишній передостанній елемент: 1азІ:=pгeyioиз

Приклад 10.5

Розробимо програму обробки впорядкованого за алфавітом списку слів. Вважаємо, що користувач може додавати та вилучати слова зі списку і при цьому список має залишатися впорядкованим. Вилучення відбувається так: користувач вводить

деяке слово, а програма шукає це слово у списку. Якщо слово знайдене, програма його вилучає, якщо ні - видає повідомлення про безрезультатність пошуку.

Алгоритм роботи з алфавітним переліком слів

1. Вважати список порожнім.
2. Вивести меню для роботи зі списком.
3. Якщо натиснута клавіша I, додати елемент до списку.
   3.1. Виділити пам'ять для нового елемента.
   3.2. Ввести нове слово та ініціалізувати ним поле даних нового елемента.
   3.3. Якщо список порожній, вважати щойно утворений елемент списком.
   3.4. Якщо список непорожній, визначити місце розташування нового елемента та вставити його до списку.
4. Якщо натиснута клавіша O, видалити елемент зі списку.
   4.1. Ввести слово, що видаляється.
   4.2. Якщо список порожній, вивести відповідне повідомлення.
   4.3. Якщо список непорожній, проглядати значення елементів списку доти, доки введене слово не буде знайдено або доки список не буде вичерпано.
   4.4. Якщо елемент із введеним значенням поля даних було знайдено, то його слід видалити.
   4.5. Якщо введене слово не збігається зі значенням інформаційного поля жодного елемента списку, вивести повідомлення про відсутність шуканого елемента у списку.
5. Якщо натиснута клавіша <3, вийти з програми.

Програма ex10_5 реалізує розглянутий алгоритм. Крім процедур вставки та видалення елементів, на увагу в цій програмі заслуговують функція ЗеагсііРіасе-Oeieie, що виконує пошук введеного слова у списку, та процедура ЗеагсііРІ асеїпзегі:, яка відшукує позицію для вставки нового елемента. В обох цих підпрограмах здійснюється послідовний перебір елементів списку, під час якого шляхом циклічного виконання присвоєння сиггепі; :=сиггепі^A. пехї: змінюється значення покажчика сиггепі. Оскільки елемент, на який посилається покажчик ргеуіоиз, має бути попереднім для елемента, що на нього посилається сиггепі, то перш ніж змінна сиггепі набуде нового значення, її старе значення має бути збережене у змінній ргеуіоиз.

```
ргодгат ex10_5;
   {створити впорядкований список, передбачити операції
    додавання, видалення та виведення елементів на екран}
изез CrI;
Type
   рїг="Їїеш;          {тип покажчика на елемент      }
   Їіеп=гесогсі        {тип елемента списку           }
        сіаіаїзігіпд;  {інформаційне поле             }
        пехі:ріг;      {покажчик на наступний елемент }
      епсі:
```

```
var Heacl: pir;          {покажчик на початок списку    }
    pempir,              {покажчик на елемент, що вводиться}
    siggepi,             {покажчик на поточний елемент  }
    pgeyioiz: pir;       {покажчик на попередній елемент }
    sij: sijag;          {код дії над елементом списку  }
    zig: zirind;         {значення елемента, що вводиться }
    Nad: Booyean;        {ознака успішності пошуку      }
{ - - - - створення одноелементного списку ==========}
procedure CrealePirstIlet;
begin
  Heacl:=pempIr;         {покажчик на вершину списку
                          перемістити на перший елемент }
  HeacГ.pexI=nil;        {покажчик на наступний елемент
                          має значення nil!             }
end;
{========= вставка елемента на початок списку ==========}
procedure IzegInVediinpind;
begin
  pempIr^.pexI=beacl;    {зв'язати новий елемент
                          з вершиною списку             }
  beacI=pempIr;          {перемістити вершину списку
                          на елемент, що додається      }
end;
{=============== пошук позиції вставки ===========}
procedure ZeargijPlaceInzegI;
begin
                         {ПОТОЧНИЙ покажчик на початок списку}
  siggepi :=Heacl:
  repeat
                         {покажчик на попередній елемент}
    pgeyioiz=siggepI;
                         {покажчик на наступний елемент }
    siggepi:=siggepI^.pexij;
    if siggepI=nil then  {якщо кінець списку            }
      find:=true         {ознака закінчення пошуку      }
    else                 {якщо список не закінчився     }
      find:=siggepI^.cial>=5Ir; {ознака продовження
                                 пошуку                 }
  until find;
end;
{========= вставка елемента в середину списку ==========}
procedure IzeInIoMicIcLe:
begin
  pgeyioiz^.pexij:=pe«pIr; {зв'язок попереднього       }
                           {і нового елементів         }
  pempIr^.nexI:=siggenI;   {зв'язок нового             }
                           {і наступного елементів     }
end:
|============ додавання слова до списку ==============}
procedure izegi;
```

```
Ьеді п
   мгііе('іприи еіетепі:');
   геасПпСзїг);              {ввести значення елемента    }
   пем(пемріг);              {виділити пам'ять для елемента}
   пемріг". ciaia :=зіг;     {занести значення в пам'ять  }
   IT Iieac=nil THen         {якщо список порожній,       }
      CreaiePirgziIiet       {формувати перший елемент
                               списку                     }
    eise                     {якщо у списку є елементи,   }
     ii зїг<Heacї". ciaia    {якщо значення нового елемента
                               менше за значення першого елемента}
       ibeп IБегіІпВеді ппіпд {вставити елемент
                               на початок списку          }
       eise                  {якщо значення нового елемента
                               більше за значення першого елемента}
       Ьеді п
         ЗеагсЬРIaceIпзегі;  {пошук місця вставки        }
         IпзегіІпioMicicЛе;  {вставка елемента           }
       enci
enci;
{========== видалення першого елемента списку ==========}
procesiure OeIPirgzi;
Ьеді п
  Iieac1=сиггепї".пехї;      {вершину списку перемістити
                               на наступний елемент       }
  cIizrozeCсиггеп);          {ЗВІЛЬНИТИ пам'ять          }
enci;
|============== пошук елемента для видалення ===========}
iипсiion ЗеагсЬРIaceOe!еге:Ьооieап;
Ьеді п
  repeai
    prеуioиз:=сиггепі;;      {переміщення покажчика від }
    сиггепі=сиггепї".пехі    {попереднього до поточного
                               елемента, доки не буде
                               знайдено потрібний елемент
                               або кінець списку          }
  иntiii (сиггепї".ciaia=зіг) or (сиггепї".пехі=пі!);
  ЗеагсїPI aceOeI eie:=сиггепї". ciaia=зїг:
enci;
{==-===== видалення елемента з середини списку =========}
procesiure OeIMicIcЛе;
Ьеді п
  prеуi оиз".пехі=сиггепї".пехі;
  cИзрозе(сиггеп);                    {ЗВІЛЬНИТИ пам'ять}
enci;
{============= видалення слова зі списку ===============}
procesiure сієї еіе;
Ьеді п
  ii Iieac1=nil iiieп          {якщо список порожній,     }
                               {вивести повідомлення      }
```

```
  Ьеdіп
    МПІЄІПСИЗІ із etply.Prezз  EMTEP...');
    reacii n;
  enci
  eize                          {якщо список має елементи
  Ьеdіп
    игіle('inpui yaiue:');
    reaciinCэIr);               {ввести слово, що видаляється
    cirreni:=Ueaci;             {покажчик на початок списку
    i i cirreni". ciala=эIr     {шуканий елемент є першим
      IHen OeIPirэi             {видалити перший елемент
      eize                      {шуканий елемент не перший
        i i 3earcHPlaceOelele   {якщо слово знайдено,
          iiien PelMicicПe      {видалити елемент
                                 із середини списку
          eize                  {якщо слово не знайдено,
          Ьеdіп                 {вивести повідомлення
            мгiIe(зIг,' пої iоипсі in 1i 3I.');
            мгiIe!n(Ргезз ЕИТЕР,..');
            геасіi n:
          enci;
    encl;
encl;
{—                      виведення списку
procesciire ouiiiзi;
Ьеdіп
  cirreni:=lieaci;              {початок списку          }
  IT cirrenI=ni1 liien          {якщо список порожній,   }
     mrile1n('l_iз1 із etpiy')  {вивести повідомлення    }
    eize                        {якщо список має елементи }
     Ьеdіп
       mriieInC'ouipui liзi:  );
       repeai                   {вивести значення елемента}
         mrile(cirrenI".clala.
          сиггепі=сиггепГ.пехі; {перемістити покажчик
                                   на наступний елемент }
       иpiii  сиггепІ=ni1;      {доки не закінчиться список}
     encl;
    мгiїein;
encl;
                    основна програма
Ьеdіп
  Heaci:=nП;                {СПИСОК порожній
  cirзcr;
  repeai
    ouiiiзi;                {вивести список
    мгiїein;                {меню програми
    мгiїein;
    mrile!n('enier coшшanci: i- inpui  ci- cieieie  д- диii');
    cЬ;=reacikey;
    cазе cii oi
```

```
      'I [1]: iпзегї;          {вставити елемент у список
      'ci': cieieie:           (видалити елемент зі списку
   encs;
 ипїП cЬ='я':                  {вийти з програми
encs.
```

!- •їх;

:I-I5I I5 епріу

```
eniег сошчапй: i- iприи   Й- Йeieie      циii
iприи eieiчепігзип
оиірии 1I5I: 5ип

eniег сошчапй: i- iприи   Й- йeieie   Ч" чиii
iiприи e1eiчenI:eaї'1(i
оиірии 1I5I: eагIЬ 5ип

eniег соттапй: i- iприи      йeieie      циii
iприи eieпепі:таг5
оиірии 1I5I: eагIН тагз 5ип

eniег сошчапй: i- iприи   й- йeieie      ^иii
iприи иаIие:паг5
оиірии 1I5I: eагIП 5ип

eniег сотпапй: i- iприи      йeieie      чиii
```

РИС. 10.13. Результати роботи програми ex10_5. Обробка списку слів, впорядкованого за алфавітом

10.3. Дерева

Розглянуті у розділі 10.2 списки, стеки та черги належать до лінійних динамічних структур даних. Визначальною характеристикою лінійних структур є те, що зв'язок між їхніми компонентами описується в термінах «попередній-наступний», тобто для кожного компонента лінійної структури визначено не більше одного попереднього та не більше одного наступного компонента. Деревоподібна структура даних, або дерево, є нелінійною структурою, що зображує ієрархічні зв'язки типу «предок-нащадок»: компонент-предок може мати багато нащадків, хоча для кожного компонента-наїцадка визначено не більше одного предка.

10.3.1. Основні поняття

Згідно з рекурсивним означенням, *дерево* з базовим типом *T* — це або порожня структура, або вузол типу *T,* з яким зв'язана скінченна кількість дерев із базовим типом Г, що їх називають *піддеревами* [5]. Із означення випливає, що піддерева (гілки) будь-якого вузла не перетинаються, тобто не мають спільних вузлів. Ця властивість розглядуваних структур і споріднює їх із справжніми деревами, адже гілки дерева «зростатися» не можуть. Наведемо графічне зображення дерева.

Вузол дерева, який не має предків, називається *коренем.* Серед будь-якої пари безпосередньо зв'язаних вузлів дерева можна виділити *предка* та *нащадка.* Нащадок є коренем певного піддерева предка. Вважається, що корінь дерева розташований на першому рівні. Кожний вузол-нащадок рівня *к* має предка на рівні *к* - 1. Максимальний рівень дерева називається його *глибиною* або *висотою.* Наприклад, на рис. 10.14 зображено дерево, глибина якого дорівнює трьом. Вузол дерева, який не має нащадків, називається *листком.* Кількість безпосередніх нащадків вузла називається його *степенем.* Максимальний степінь вузла у певному дереві називається *степенем дерева.*

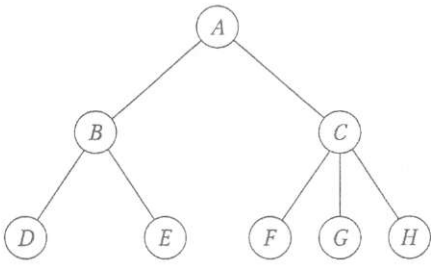
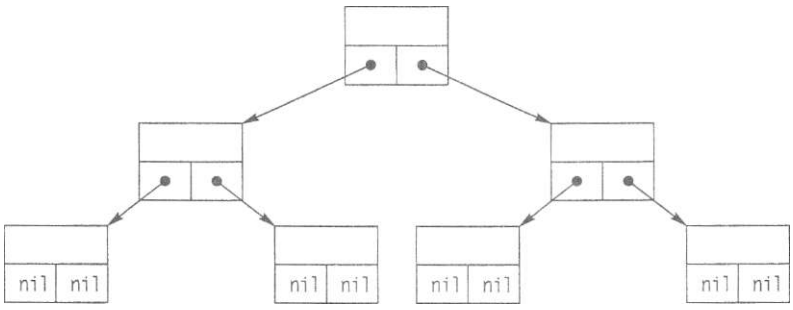


Рис. 10.14. Дерево

У даному підручнику розглядаються лише *бінарні дерева,* тобто дерева, степінь яких не перевищує двох. Отже, будь-який вузол бінарного дерева може бути зв'язаним не більше ніж із двома піддеревами, що називаються *лівим* і *правим* піддеревами вузла. Оголосимо тип вузла бінарного дерева на мові Разсаі:

```
Type  piг^ociе;                {тип покажчика на вузол дерева}
      Иociе=record1            {тип вузла дерева          }
            ciaia:зігіпд:      {нформаційне поле вузла    }
            Iеії, гідНі:piг;   {покажчики на ліве та
                                  праве піддерево         }
      enci;
```

Для листків покажчики Іеіі та гідіїї мають значення пії. Приклад бінарного дерева як структури даних наведено на рис. 10.15.



Рис, 10.15. Дерево як структура даних

10.3.2. Алгоритми роботи з бінарними деревами

Найпоширенішими операціями під час роботи з деревами є: створення дерева, включення вузла в дерево, видалення вузла з дерева, обхід дерева та пошук у ньому. Розглянемо алгоритми цих операцій детальніше на прикладі бінарних дерев.

Створення бінарного дерева

Найпростіший спосіб побудови бінарного дерева полягає у створенні дерева симетричної структури із наперед відомою кількістю вузлів. Усі вузли-нащадки, що створюються, рівномірно розподіляються зліва та справа від кожного вузла-предка. При цьому досягається мінімально можлива глибина для заданої кількості вузлів дерева. Правило рівномірного розподілу *n* вузлів можна визначити рекурсивно:

- перший вузол вважати коренем дерева;
- створити ліве піддерево з кількістю вузлів *nie/ї = n* сшу 2;
- створити праве піддерево з кількістю вузлів *nгі§Hг = n - nie/i -* 1.

Приклад 10.6

Створимо бінарне дерево із заданою користувачем кількістю вузлів. Тип вузла (компонента) дерева було оголошено у розділі 10.3.1. Оскільки структура дерева визначена рекурсивно, то для його створення та відображення можна розробити рекурсивні підпрограми.

Функція створення дерева їгее отримує один цілочисловий параметр Атоипі-Мосіе, що визначає кількість вузлів дерева та повертає покажчик на його корінь. Якщо кількість вузлів дорівнює нулю, дерево порожнє, а отже, виконано умову завершення рекурсії і функція має повернути значення пі 1. Якщо кількість вузлів дерева відрізняється від нуля, необхідно виділити пам'ять для кореня дерева, за наведеними вище формулами обчислити кількість вузлів у лівому та правому піддеревах і двічі рекурсивно викликати функцію їгее для створення піддерев. Для посилання на корінь дерева використаємо локальний покажчик пемюсіе. Значення покажчиків на корені піддерев, що їх повертатиме функція їгее в результаті рекурсивних викликів, слід присвоїти полям Іеії та гідіїї змінної пемпосіе".

Дерево відображатиме рекурсивна процедура ргі гтЬігее. Піддерево рівня І виводитиметься так: спочатку буде відображене ліве піддерево, потім - корінь піддерева рівня І, перед яким буде виведено І пробілів, далі - праве піддерево. Приклад створюваного програмою ех10_6 дерева зображено на рис. 10.16 і 10.17. Дерева такого типу називають *синтаксичними деревами* арифметичних виразів.

```
progrдат ех10_6; {створення та відображення бінарного дерева}
Type  рїг=посіе:              {тип покажчика на вузол дерева
      посіе=record            {тип вузла дерева
              сіаіа:$їгіпд;   {інформаційне поле вузла
              Іеії.           {покажчик на ліве піддерево
              гіділі::ріг;    {покажчик на праве піддерево
            end;
уаг  п:іпіедег;               {кількість вузлів дерева
     гооїїрїг;                {покажчик на корінь дерева
```

```
{============== створення бінарного дерева ===============}
iипciion TreeCAшoипiMocIe:iпieдer):piг;
{AтoипiMocie - кількість вузлів дерева,
 piг - покажчик на корінь дерева}
маг пемпосіе:piг;                {покажчик на новий вузол   }
    IeШocie3,                    {кількість вузлів у лівому }
    КідПШcie3iiпieдer;           {і правому піддеревах      }
    зig:sirinд:                  {інформаційне поле вузла   }
Ьеgiп
  if АптаипMocIe =0 iЬen         {якщо вузлів немає,        }
    iree:=пil                    {дерево порожнє            }
  eise
  Ьеgiп
    1_eШocie5: = Ашoипîocie ciN 2; {кількість вузлів
                                    у піддеревах           }
    Кi дИMocIe3: = AтoипMocie - i-eШocie3 -1;
    mgiieC Eпieг посѕ ciaia:');
    reaciїn(зig):
    Мем(пемпосіе);     {ВИДІЛИТИ пам'ять для нового вузла}
    у/їїН пемпосіе^ cio
    Ьеgiп
      ciaia:=зїг;          {задати інформаційне поле вузла }
                           {створити ліве піддерево        }
      1eii:=Tree(1_eiiMocie5):
                           {створити праве піддерево       }
      гідНi :=Tree(РідЬiMocIe3);
    encs:
    Tree:=пемпосіе:        {значення, що повертається      }
  encs:
encs;
{================= відображення дерева ====================}
procesciиre Priniigee(KooiTree:piг;L:iпieдer);
{KooiTree - покажчик на корінь дерева: I - номер рівня     }
yar i:iпїедer;                           {параметр циклу}
Ьеgiп
  ii PooiTree◇пil iЬen           {дерево не порожнє         }
    miiЬ KooiTree^ cio
    Ьеgiп
      PriniigeeC1eii,Ь+1);     {вивести ліве піддерево     }
      ior i :=i io I. cio mgiie(' ');
      mgiie1n(ciaia);          {вивести значення вузла     }
      PriniigeeCгідНiЛ+1);     {вивести праве піддерево    }
    encs;
encs;
|=================== основна програма ====================}
Ьеgiп
  mgiie!n(Creaie апci ciizpiay igee');
  mgiie!n('Eпieг питЬег oi igee''з посіез');
  reaciin(n):              {задати кількість вузлів дерева }
  rooi:=Tree(n);           {створити дерево                }
  mgiie!n(CreaieciigeeI');
  Priniigee(rooi.O);       {відобразити дерево             }
  reacl' n;
encs.
```

```
ОЬРЧІІМ\тП п 1Ж
Creaie апй Йізріау їгее
Enier питЬег оЕ Iree'5 пойез
7
Enier пойе йаіа:*
Enier пойе йаіа:-
Enier пойє йаІа:й
Enier пойе йа1а:с
Enier пойе йа1а:+
Enier пойе йаІа:Ь
Enier пойе йа1а:а
Creaiей Iree
        й
```

.Ж

Рис. 10.16. Результати роботи програми ex10_6. Створення та відображення бінарного дерева

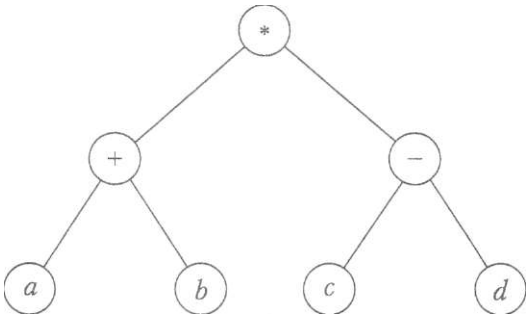


Рис. 10.17. Бінарне дерево, створене програмою ex10_6

Обхід дерева

Алгоритм доступу до всіх вузлів дерева називається *обходом дерева.* Трьома основними способами обходу дерева є обхід зверху вниз, зліва направо та знизу вверх. Якщо при обході дерева, що зображене на рис. 10.17, ми виписуватимемо значення його вузлів у тому порядку, в якому вони трапляються під час обходу, то отримаємо послідовності символів, наведені в табл. 10.1.

Таблиця 10.1. Результати обходу дерева

Спосіб обходу	Послідовність символів
Зверху вниз	* + *аЬ -с й*
Зліва направо	*а* + *Ь** с -*сі*
Знизу вверх	*а Ь* + с *(1* -*

У результаті обходу синтаксичного дерева зверху вниз утворюється *префіксна* форма виразу, при обході знизу вверх — *постфіксна* форма, а при обході зліва направо — *інфіксна* форма. Будь-який спосіб обходу дерева можна реалізувати рекурсивною процедурою. Такі процедури наведені у прикладі 10.7.

Приклад 10.7

Розробимо три рекурсивні процедури обходу бінарного дерева. До цих процедур передається параметр-значення, що є покажчиком на корінь дерева. Тіло всіх трьох процедур містить однаковий набір операторів. Першою виконується перевірка того, чи не є дерево порожнім. Якщо дерево порожнє, здійснюється рекурсивне повернення, а в іншому разі виводиться значення вузла і рекурсивно викликаються процедури обходу для лівого та правого піддерев. Порядок цих трьох операторів і визначає форму виразу, що буде створений у результаті обходу. А саме, якщо виведення значення вузла виконуватиметься першим, то буде отримано префіксну форму виразу, якщо другим — інфіксну, а якщо третім - постфіксну форму. Наведемо коди процедур обходу дерева.

```
{============== обхід дерева зверху вниз ==============}
procedure Preii xorcier(PooïTree: pir);
begin
  ii KooiTreeonП ithen
  begin
    mriie(Kooi:Tree^л.ciaia,' '):
    Preiixorcier(KooITree^A.Leii);
    Preiixorcier(PooiTree^л. gidiïi);
  end;
end;
{============ обхід дерева знизу вверх ================}
procedure Pobiixorcier(PooiTree:pïr);
begin
  ii KooiTreeonii Ithen
  begin
    Poziii xorcier (PooïTree^A. I e ii);
    Pozïiixorcier(KooïTree^л. gidIi);
    mrïïe(Poo1:Tree^л.cIa1:a,' ');
  end;
end;
{=============0хід дерева зліва направо ==============}
procedure InTixorcIerCPooiTreefïr);
begin
  iT PooïTreeonП IHen
  begin
    InTi xorcier(PooiTree^. IeTI);
    mrii:e(KooiTree^A.cIa1:a,' ');
    InPixorcier(PooITree^л.gidI1);
  end;
end;
```

В основній програмі обходу дерева необхідно здійснити виклик щойно записаних процедур після виклику функції створення дерева. Результати роботи програми, у якій створюється дерево та виконується його обхід, зображено на рис. 10.18.

```
{основна програма обходу дерева}
Ьеділ
  мгїїе1п(СгеаТе апсі сіізріау Тгее');
  мгїїе1п(ЕпТег питЬег оТ Тгеез посіез');
  геасПп(п);                  {ввести кількість вузлів}
  гооТ:=Тгее(п);              {створити  дерево       }
  мгіТе!п(СгеаТесі Тгее');
  РгіпТТгее(гооТ.О);          {вивести  дерево        }
  мгіТеїп;
  мгіТе1п('ргеТіхогсіег Тгауегзаї'):
  РгеТіхогсіег(гооТ);         {обхід  зверху  вниз    }
  мгіТеїп:
  мгіТе!п('розТТіхогсіег Тгауегзаї');
  РозТТіхогсіег(гооТ);        {обхід  знизу  вверх    }
  мгіТеїп;
  мгіТе!п(іггГіхогсіег Тгауегзаї');
  ІпТіхогсіег(гооТ);          {обхід  зліва  направо  }
  мгіТеїп;
  геасІ п;
епсі.
```

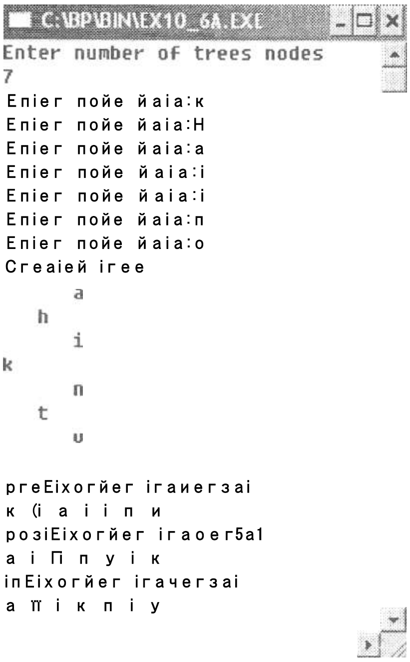


Рис. 10.18. Результати роботи програма що здійснює обхід дерева

Дерева бінарного пошуку

Бінарні дерева часто використовуються для зображення множин, елементи яких потрібно знаходити за заданим значенням ключа. Такі дерева називаються *деревами бінарного пошуку.* Особливість подібного дерева полягає в тому, що для будь-якого його вузла x значення всіх вузлів лівого піддерева *x* не більші за значення *x,* а значення всіх вузлів правого піддерева *x* не менші за значення *x.* Така властивість називається *впорядкованістю ключів у дереві.*

Вузол із заданим значенням ключа буде знайдений доволі швидко, якщо спускатися від кореня дерева бінарного пошуку за таким правилом: ліве піддерево обирається тоді, коли значення розглядуваного вузла більше за шукане, а праве піддерево - тоді, коли вказане значення менше за шукане. Якщо значення вузла дорівнює шуканому, пошук слід завершити. Якщо пошук привів до порожньої гілки дерева, то ключового значення в дереві немає. Легко зрозуміти, що для знаходження за таким алгоритмом значення ключа серед и-елементної множини буде переглянуто O(1o§ *n)* елементів.

Приклад 10.8

Розробимо функцію знаходження ключового значення в бінарному дереві пошуку. До функції передається ключове значення і покажчик на корінь дерева, а повертає функція покажчик на вузол, значення якого дорівнює шуканому. Як ознаку успішності пошуку використаємо логічну змінну ТоипсI. Пошук починається з кореня дерева і триває доти, доки шукане значення не буде знайдене або доки всі вузли дерева не будуть перевірені.

```
ТипсTion 1ocaTion(key:Siing;KooTTree:pTr):pTr;
                  {key - ключове значення          }
                  {KooTTree - покажчик на шуканий вузол }
yar ToипсI:Booïean:          {ознака успішності пошуку}
Ьеgin
  ToипсI :=Ta1$e;
  мЬПe (KooTTree<>nil) and (noT ToипсI) do
  {цикл триває доти, доки і листа не досягнуто,
   і ключового значення не знайдено}
  Ьеgin
    IF KooTTree^.ciaTa=key    {перевірити значення вузла}
      Tьen ToипсI=Tгие        {значення знайдено        }
      eise IF KooTTree^.ciaTa>key {значення не знайдено}
              Tьen KooTTree=KooTTree^.IeTT
              eise KooTTree:=KooTTree^.rigIT;
  end;
  1ocaTion=KooTTree;
end;
```

В основній програмі здійснюється **виклик** функції пошуку потрібного значення вузла дерева. Щоб повторити пошук вузла, необхідно використати оператор циклу, який припинить свою роботу після натискання клавіші N.

```
уаг п:іпТедег;                {кількість вузлів дерева }
    гооїːріг;                 {покажчик на корінь дерева}

    кеуТоипсі5Tiitp;          {шукане значення вузла   }
    гооТТоипсі ріг:           {покажчик на шуканий вузол}
    сИісітаг;                 {код натиснутої клавіші  }
Ьеді п
  «гіТе1п(СгеаТе апсі сіізріау їгее');
  мгіТе!п(ЕпТег питЬег оТ їгее' з посіез');
  геасііп(п);                       {ввести кількість вузлів}
  гоої=Тгее(п);                     {створити бінарне дерево}
  гереаі;
    мгіТе1п(ЕпТег кеу Іо зеагсЬ');
    геасіїп(кеуТоипсі);             {ввести ключ пошуку}
    гооТТоипсі=ЬосаТіоп(кеуТоипсІгооТ); {виконати пошук}
    і гооТТоипсі<>nil               {якщо вузол знайдено}
      Тьеп «гіТе1п('ТоипсІ:' .гооТТоипсГ.сІаТа)
      eise мгіТеїпСпоТ ТоипсІ);
    мгіТе1п(сопїіпие?у/п');  {запит на продовження  }
    сіі=геасікеу:            {пошуку                 }
  ипТіі сіі='п';
епсі.
```

```
7
ЕпТег пойе йаТа:к
ЕпТег пойесіаІа:м
ЕпТег пойе йаГ'а:а
ЕпТег пойе йаТа:і
ЕпТег пойе йаТа:&
ЕпТег пойе йа(:а:п
ЕпТег пойе йаЕаіц
ЕпТег кеу Ео зеагсй
I
Еоипй:і
сопТіпие?у/п
ЕпТег кеу То зеагсїт
Т
Еоипй:т
ісопТ1пие?у/п
ЕпТег кеу То зеагсїт
п
&оипй:п
сопТіпие?у/п
ЕпТег кеу То ееагсїт
И
поТ Еоипй
сопТіпие?у/п
```

Рис. 10.19. Результати роботи програми, що здійснює пошук ключового значення у бінарному дереві пошуку

Включення вузлів у дерево

Перейдемо до розгляду дерев змінної структури, тобто таких дерев, кількість елементів яких під час роботи певної програми може збільшуватися або зменшуватися. Дерева змінної структури можна використовувати, зокрема, при створенні частотного словника. Задача створення частотного словника формулюється так: потрібно визначити, скільки разів зустрічається кожне слово в заданій послідовності слів.

Послідовність слів можна зобразити у вигляді дерева бінарного пошуку. В інформаційних полях вузла такого дерева зберігатиметься слово і кількість його повторень. У разі виявлення в тексті чергового слова дерево переглядають, починаючи з кореневого вузла. Якщо слово в дереві знайдене, то лічильник його повторень збільшується. Але якщо слово в дереві не знайдене, воно включається в дерево із значенням лічильника, що дорівнює 1. Дану задачу називають іще *пошуком із включенням.*

Включення вузла в дерево має здійснюватися так, щоб не порушувалася властивість упорядкованості ключів. Це правило буде дотримане, якщо застосувати розглянутий вище алгоритм знаходження ключового значення у бінарному дереві пошуку, а включення нового елемента здійснювати тоді, коли пошук завершився безуспішно. У прикладі 10.9 буде реалізований рекурсивний варіант алгоритму пошуку із включенням.

Приклад 10.9_

У програмі, що реалізує пошук із включенням, компоненти бінарного дерева містять два інформаційні поля: поле рядкового типу для збереження слова та цілочислове поле для збереження кількості повторень слова. Наведемо оголошення типу такого компонента разом із оголошеннями використаних у програмі глобальних змінних:

```
Type pIr=^nocIe;
     nocIe=record
             ciaia: zigi pd; {значення, що включається в дерево}
             coupi: 1 grbeder:   {кількість повторень значень}
             Ieii, gidni: pir;   {покажчики на ліве та праве }
         end;                    {піддерева                  }
var  n:inIeder;                  {кількість вузлів дерева    }
    rooI:pIr;                    {покажчик на корінь дерева  }
    5eargcHkey:Igipd;            {шукане слово               }
    cH:cHar;   {символ, що позначає кінець процесу пошуку}
```

Запишемо код процедури ZeargcПIpzeП, призначеної для пошуку та включення вузла у бінарне дерево, і код основної програми. Шукане слово передаватимемо до згаданої процедури через параметр-значення key, а покажчик на корінь дерева - через параметр-змінну KooITree. Після того як роботу процедури ZeargcH-Ipzegi буде завершено, покажчик KooITree посилатиметься на знайдений вузол. На рис. 10.20 зображено результати роботи програми пошуку із включенням. Як бачите, етап введення даних на ньому не показано.

```
{=========== пошук і включення вузла в дерево ==========}
procedure SearchInsertKey(Key: Integer; var Root: Tree; Found: Boolean);
{key - ключ пошуку. Root - покажчик на вузол }
begin
  if Root = nil
  then                    {покажчик порожній, слово не знайдене}
    begin
      new(Root);{виділити пам'ять для нового вузла }
      with Root^ do
        begin
          data := key;    {записати у вузол шукане слово }
          count := 1;     {задати кількість його повторень}
          left := nil;    {піддерева даного вузла        }
          right := nil;   {зробити порожніми             }
        end;
    end
  else                    {пошук слова                   }
    if key < Root^.data   {пошук у лівому піддереві }
      then SearchInsertKey(Key, Root^.left, Found)
      else
        if key > Root^.data {пошук у правому піддереві}
          then SearchInsertKey(Key, Root^.right, Found)
          else                {слово знайдене          }
            begin
              Found := true;
              Root^.count := Root^.count + 1;
            end;
end;
{================= основна програма ===================}
begin
  writeln('Create a binary search tree');
  writeln('Enter a set of integers');
  readln(n);                {ввести кількість вузлів дерева}
  Tree := nil;              {створити дерево                }
  writeln('Create a tree');
  PrintTree(Tree, 0);       {відобразити дерево             }
  writeln;
  repeat
    write('Enter key for search and insert');
    readln(SearchKey);      {ввести слово для включення   }
    SearchInsert(SearchKey, Tree, Found); {пошук із включенням}
    writeln('count = ', Found);
    Ch := SearchKey;
  until Ch = '*';
  PrintTree(Tree, 0);       {вивести дерево після включення}
  readln;
end.
```

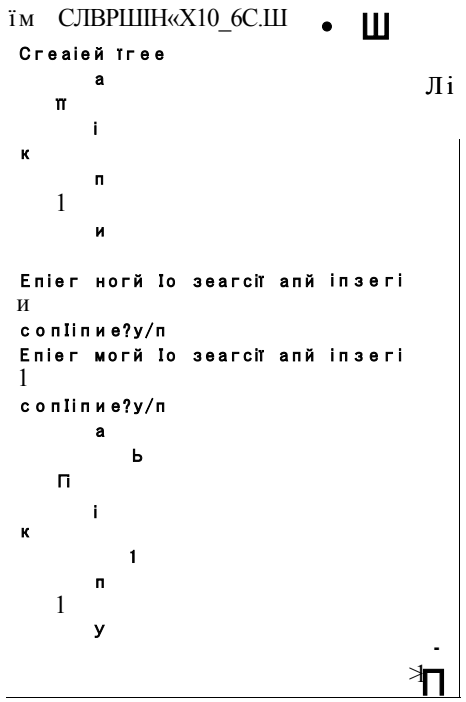


Рис. 10.20. Результати роботи програми, що здійснює пошук із включенням

Видалення вузлів бінарного дерева

Розглянемо задачу видалення вузла бінарного дерева пошуку. Можливі три випадки: вузол, що видаляється, не має нащадків, тобто є листком дерева; вузол, що видаляється, має одного нащадка; вузол, що видаляється, має двох нащадків.

Задача розв'язується найпростішим способом тоді, коли вузол, що видаляється, є листком дерева. У такому разі слід звільнити ділянку динамічної пам'яті, яку цей вузол займав, та присвоїти значення пі І покажчикові на даний вузол.

Якщо видаляється вузол *х,* який має одного нащадка, то покажчику на цей вузол слід присвоїти адресу його нащадка і звільнити пам'ять, яку вузол *х* займав. Операцію видалення вузла, що має одного нащадка, проілюстровано на рис, 10.21.

Найскладнішим є випадок, коли вузол *х,* що видаляється, має двох нащадків. У цьому разі на місце *х* слід переставити інший вузол дерева так, щоб не порушувалася властивість впорядкованості ключів. Вузол, що переставляється, називається *термінальним.* Один із способів визначення термінального вузла полягає у виконанні спуску по правій гілці лівого піддерева *х* доти, доки не буде знайдено вузла без правого нащадка. Цей вузол і є термінальним. Справді, значення цього вузла не менше за значення всіх вузлів лівого піддерева *х,* оскільки він належить правій гілці цього піддерева і не має правого нащадка. З іншого боку, значення

знайденого в такий спосіб вузла не більше за значення всіх вузлів правого піддерева *x,* оскільки він належить лівому піддереву *x.* А отже, значення знайденого вузла може бути записане до вузлах без порушення впорядкованості ключів. Сам термінальний вузол має бути видалений після копіювання його значення до вузла *x.* Оскільки термінальний вузол має одного нащадка, його видалення є доволі простою операцією, що розглядалася вище. Дерево до і після видалення вузла Д який має двох нащадків, зображено на рис. 10.22.

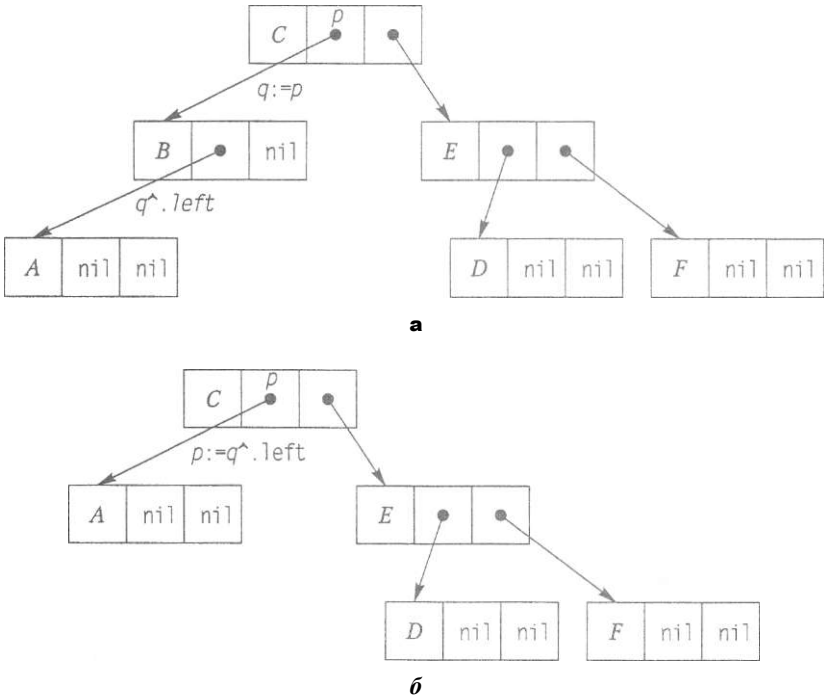


Рис. 10.21. Бінарне дерево до (а) і після (б) видалення вузла з одним нащадком

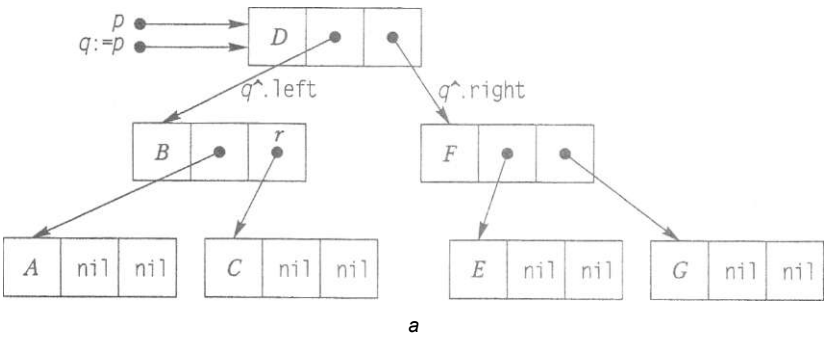


Рис. 10.22. Бінарне дерево до видалення вузла, що має двох нащадків (а)

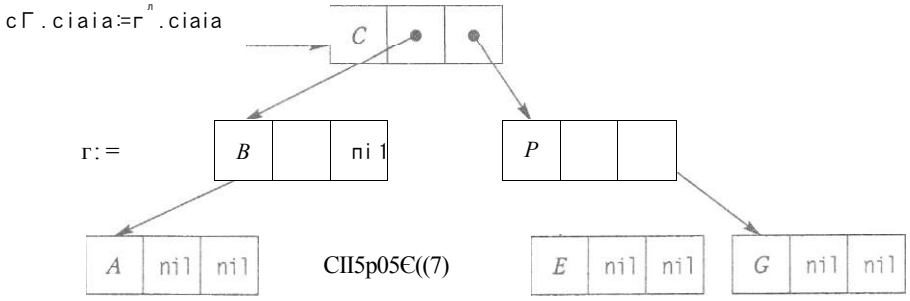


Рис. 10.22 (*продовження*): бінарне дерево після видалення вузла, що має двох нащадків (б)

Приклад 10.10_

Розробимо процедуру 5еагс№elele, що знаходить за заданим ключем вузол бінарного дерева пошуку і видаляє його. Запишемо також програму ex10_7, у якій ця процедура використовується. Усередині процедури ЗеагсїїOeIeIe оголосимо локальну рекурсивну процедуру СазеТмоСИі 1 сі, яка опрацьовуватиме той випадок, коли вузол, що видаляється, має два нащадки. Процедура СазеТмоСИїсі матиме параметр-змінну геТ, який на певному рівні рекурсії набуватиме значення покажчика на термінальний вузол. Коли термінальний вузол буде знайдено, його значення скопіюється до вузла сієї Mocle^A, який видаляється, а покажчику геТ присвоїться адреса лівого нащадка термінального вузла. Функцію Igee та процедуру prinIigee у програмі ex10_7 не наводитимемо, їх можна запозичити з програми ex10_6. Результати роботи програми, без етапу введення даних, зображено на рис. 10.23.

```
prograt ex10_7;                              {видалення вузла}
 izes cgi;
Type pIr=^nocie;
     nocie=recorci
         ciaIa:zIgipd; {значення, що включається в дерево}
         1eII,gidiiI:pIr:      {покажчики на ліве та праве
     encci;                    {піддерева
yar n:inIedeg;                 {кількість вузлів дерева
     gooi:pig;                 {покажчик на корінь дерева }
     zeagciikeyZgipd;          {шукане слово              }
     cii:ciiag; {символ, що позначає кінець процесу пошуку }

         функція Igee з програми ex!0_6

         процедура PrginITgee з програми ex!0_6

              видалити задании вузол
procecivre Zeagcii0e1e1e(key:Siring; yar pNo.cie:pIr);
{key - ключ пошуку,
pNo.cie - покажчик на вузол, що видаляється}
yar cieї Mocie: pig;   {покажчик на вузол, що видаляється}
```

```
{          пошук листка 1 переміщення його вмісту
           у вузол, що видаляється. . . . . . . . . . . . . .}
ргосесиге СазеТдаСНі 1 й{уаг геї:рїг);
                    {геТ - покажчик на термінальний вузол}
Ьеді п
    іТ геТ".гідНТ <>п11          {спуск по правій гілці}
      ТИеп СазеТмоСНі 1 сіСгет^.гідИіс)
      еізе
      Ьедіп {заміна вузла, що видаляється, термінальним}
        йеlМосІе".сїаїа: = геТ".сіаїа;
        сіеї Мосіе :=геІ:
        геї:=геТ".1еТї;
      епсі;
  епсі;
{               тіло процедури 5еагсіїОе!еїе. . . . . . . . . . . . . .}
Ьедіп
   іТ рИосіе=піі ТНеп мгїїе1п('посіє із поі Іоипсї')
   еізе іІ кеу<рМоае".сІаїа
         ТЬеп 5еагсЬ0еіеІе(кеу. рМосіе". ІеТї)
         еізе іТ кеу>рМоОе".сІаїа
              ТНеп 5еагсіі0е1еїе(кеу, рМосіе". гідНІ)
              еізе                          {вузол знайдено}
                Ьедіп
                  сіеї Мосіе: =рМосіе;
                  ІТ сіеіМосіе". гід!іТ=пі1
                  ТНеп р^сіе :=сіе1МосІе".1еїї
                  еізе іТ сіе1Мос1е".1еїї=пі1
                       ТНеп рИосіе=ае1 Мосіе".гідіїї
                       еізе СазеТиоСііі1сі(сіе1МосІе".1еТї);
                  сіі зрозеС сіеї Мосіе);
              епсі;
  епсі;
|================== основна програма ===================}
Ьедіп
  мгїїе1п(СгеаТе. сіізріау Ігее апсі сіеіеіе посіє');
  мгїїе1п(ЕпТег пиглЬег оТ їгее''з посіез');
  геасії п(п);
  гооі:=їгеє(п);                    {створити бінарне дерево}
  мгіІе1п(Сгеаіей їгее');
  ргіпІІгее(гооІ.О);                {вивести дерево на екран}
  мгїїеіп;
  гереаї
    мгїїе1п('Епіег кеу іо зеагсії апсі сіеіеіе');
    геасПпСзеагсїїкеу);
    5еагсЬ0е1еІе(зеагсЬкеу, Кооі);         {видалити вузол}
    мгіІе!п('сопІіпие?у/п');
    сіі=геасікеу;
  ипїіі сіі=' п';
  ргіпІІгее(гооІ.О);     {вивести дерево після видалення}
  геасії п;
епсі.
```

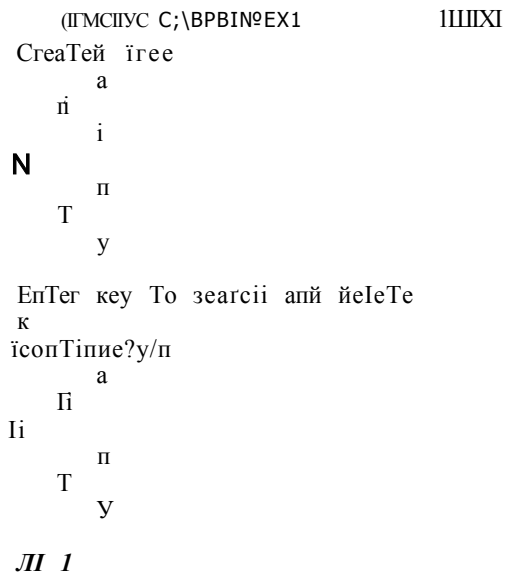


Рис. 10.23. Результати роботи програми ex10_7. Видалення вузла бінарного дерева

10.4. Масиви в динамічній пам'яті

ЯК уже зазначалось у розділі 10.2, зображення послідовностей однотипних даних у формі лінійних списків має і переваги, і недоліки. Основним недоліком є значна трудомісткість операції доступу до елемента лінійного списку за його номером. Цей недолік непритаманний масивам. Проте масиви, про які йшлося в розділі 7, мали іншу суттєву ваду — вони були статичними, тобто їх розмір визначався під час розробки програми. У даному розділі розглядаються *динамічні масиви,* розмір яких визначається під час виконання програми і доступ до елементів яких здійснюється так само швидко, як і до елементів статичних масивів.

Динамічний масив ідентифікується покажчиком на його перший елемент. Базовий тип цього покажчика в мові Разсаі оголошується у доволі специфічний, порівняно з іншими мовами програмування, спосіб. А саме, він оголошується як тип одноелементного статичного масиву, базовий тип якого збігається із базовим типом динамічного масиву. Наприклад:

```
Type агг=array[0..0]оТ inТедег;
yar сіупагг: "агг;
```

Тут сіупагг — покажчик на динамічний масив даних типу inТедег, агг - тип цього покажчика. Зауважимо, що, хоча згідно з синтаксисом покажчик сіупагг посилається на одноелементний масив, процедурою СёТМет (див. розділ 10.1.4) можна виділити для цього покажчика довільний обсяг пам'яті, який не перевищує обсягу

одного сегмента, тобто 64 Кбайт, або 65 536 байт. Тому за допомогою покажчика сіупаг можна посилатися на елементи масиву доволі великого обсягу. Наприклад:

```
беТМет(сіупагг, 1000*5i2e o T ( inТедег));
i :-3;
сіупагг"[i] :=1;
```

У цьому фрагменті коду було виділено пам'ять для динамічного масиву, що містить 1000 елементів типу **inТедег,** і присвоєно значення 1 його третьому елементу. Тип змінної і має збігатися з типом індексів масиву, який було згадано в оголошенні типу агг. Тобто типом змінної і має бути один із цілочислових типів.

УВАГА — —

Індекс елемента динамічного масиву не може бути виразом, значенням якого є константа. Наприклад, запис **сіупагг"[3]:=1 є некоректним, оскільки масив** сіупагт" **оголошено як одноелементний і** тому **при компіляції виразу сіупагг"[3] буде виявлено синтаксичну помилку Error 76: Conziari! оиі оїгапде (Константа не належить допустимому діапазону).**

А як оперувати масивами, що їх розмір перевищує 64 Кбайт? Для цього можна створити масив покажчиків. Наприклад, масивом, який складається з 100 000 елементів типу inТедег можна оперувати, виділяючи пам'ять під 10 000 елементів типу inТедег для кожного з 10 покажчиків:

```
Type агг=array[0..,0]oT inТедег;
yar p:array[0..,9]oT "агг; i:inТедег;

Tor i :=0 To 9 cio
  CeTMemKp[i],10000*5i2e0T(inТедег));
i:=150:
p[2Г[1]:-1:
```

Вираз р[2]"[i] посилається на елемент 100 000-елементного масиву з індексом 20 150 = 10 000 • 2 + *i* Взагалі, якщо кожна з частин великого масиву містить питЬег елементів, то вираз р[Л^{A}[к] посилається на елемент з індексом ,]*питЬег+к. Із цього випливає, що доступ до ***m-mo*** елемента великого масиву можна здійснити за допомогою виразу р[Т сшу **питЬег]**^{A}[Т **тосі питЬег].**

Наостанок зауважимо: аби за допомогою одного й того самого покажчика можна було посилатися на динамічні масиви різних базових типів, цей покажчик слід оголошувати нетипізованим, а при згадуванні його імені використовувати операцію перетворення типів: <ім*я типу>(ім'я покажчика). Така техніка застосовується у прикладі 10.11.

Приклад 10.11

Потрібно отримати масив з 48 000 елементів типу **IonдiпT.** Оскільки зберігання елемента даних типу 1 опді пТ потребує чотирьох байтів пам'яті, зберігання всього масиву вимагатиме 4-48 000 = 192 000 байт пам'яті. Оскільки 3 • 65 536 = 196 608, то масив може бути розташований у трьох сегментах пам'яті, тобто складатися із трьох частин, кожна з яких містить 16 000 елементів.

У програмі ex10_8 елементам великого масиву присвоюються їх порядкові номери. На екран будуть виведені значення тих елементів, номери яких кратні 4000. Результати роботи програми ex10_8 наведено на рис. 10.24.

```
prograt ex10_8;   {великі масиви в динамічній пам'яті }
consi blosk=3;    {кількість блоків по 16 000 елементів}
Type agg=array[0..0]oI Iopdipi;  {тип масиву          }
     pIg=^agg;                   {тип покажчика на масив}
yar p:array[0.,blosk]oT poipieg; {масив покажчиків     }
    pitbeg,              {кількість елементів у блоці
    Ioiai : Iopdipi ;    {загальна кількість елементі
    3 : Iopdipi:         {параметр циклу
    i: ipiedeg;          {допоміжна змінна
beqin
  pitbeg:=16000:
  Ioiai:=pitbeg*blosk:   {Ioiai = 48 000
  mgiïeInC'Igee tetogu: '.tetayaii,';tax agea:
        taxayaii);
  Ior I:=0 Io blosk-1 cio  {ВИДІЛИТИ динамічну пам'ять }
    deIshet(p[i]. pitbeg*Si2eoI(Iopdipi));
  Ior ]:=0 Io Ioiai-I cio  {записати у пам'ять значення }
    pIg(p[ ciiV pitbeg])[ toci pitbeg]:=,];
  i:=0;                  {i - номер стовпчика чисел  }
  ]:=0;
  «Hi 1 є ]<IoIa1 cio    {вивести масив              }
  beqin
    mgiIe(pIg(p[ ciiV pitbeg])"[z toci pitbeg]:8):
    i:=i+1:
    ii i=4 IIiep beqin mgiïeip; i:=0 epci;
    ]:=3+4000;
  epci;
  mgiïeip;
  mgiIe1n('Igee tetogu; ', gpeшayai 1.': tax agea: ',
        taxayaii);
  geacïï n:
epci.
```

```
OIIIP1IMI1II -IOÏ*
Eree tetogu: 343689928; tax area : 16711688
        8     4888     8888    12888
    16888    28888    24808    28888
    32888    36888    48888    44888

<=ree tetogu: 343865696; tax area: 16711688

< I
```

РИС. 10.24. Результати роботи програми ex10_8. Робота з динамічним масивом

Висновки

4 Статичні змінні характеризуються тим, що ділянки оперативної пам'яті, в яких зберігаються їх значення, визначаються на етапі компіляції програми і не змінюються під час її виконання. Значення таких змінних зберігаються в сегменті даних.

• Динамічні змінні створюються і знищуються у процесі виконання програми, їх значення зберігаються в динамічній пам'яті.

4- Доступ до динамічних змінних здійснюється за їх адресами в динамічній пам'яті. Для збереження адреси динамічної змінної використовується посилальний тип даних, а змінна посилального типу даних називається покажчиком. Значенням покажчика є адреса області пам'яті, де зберігається певний елемент даних.

4- Покажчик, що може посилатися лише на дані певного типу, називається типізованим, а відповідний тип даних — базовим.

•4 Нетипізований покажчик не зв'язується з жодним типом даних і оголошується як покажчик типу роіпіег.

+ Над покажчиками можуть виконуватися три операції: присвоєння, порівняння та розіменування. Покажчики можна порівнювати, використовуючи операції = (рівність) та о (нерівність). Покажчику можна присвоїти значення адреси статичної змінної або підпрограми, значення іншого покажчика, а також значення адреси, що його повертає функція Ріг. Щоб отримати значення, на яке посилається покажчик, потрібно виконати операцію його розіменування.

4- У мові Разсаі використовуються три методи роботи з динамічною пам'яттю: за допомогою процедур і Різрозе; СеІМеш і РгееМет; Маік і Реї еазе.

4 Структура даних називається динамічною, якщо її розмір визначається і може змінюватися під час виконання програми. Основними різновидами динамічних структур даних є лінійні списки, дерева (нелінійні списки) і динамічні масиви.

4- Лінійний список є формою зображення послідовності однотипних елементів. Елемент однозв'язного лінійного списку є записом, що містить інформаційні поля та покажчик на наступний елемент. Елемент двозв'язного списку містить інформаційні поля і покажчики на попередній і наступний елементи.

4 Стек -- це однозв'язний лінійний список, у якому елементи додаються та видаляються з його вершини, тобто з початку списку.

4 Черга - це однозв'язний лінійний список, в якому елементи додаються в кінець списку, а видаляються з його початку.

4- Дерево з базовим типом *Т* — це або порожня структура, або вузол типу *Т,* з яким зв'язана скінченна кількість дерев із базовим типом *Т,* що їх називають піддеревами.

4- Вузол дерева, який не має предків, називається коренем. Серед будь-якої пари безпосередньо зв'язаних вузлів дерева можна виділити предка та нащадка: нащадок є коренем певного піддерева предка. Вважається, що корінь дерева розташований на першому рівні. Кожний вузол-нащадок рівня ***к*** має предка на

рівні *к* - 1. Максимальний рівень дерева називається його глибиною або висотою. Вузли дерева, що не мають нащадків, називаються листками. Кількість безпосередніх нащадків вузла називається його степенем. Максимальний степінь вузла у певному дереві — це степінь дерева.

• Дерево називається бінарним, якщо кожен його вузол може бути зв'язаним не більше ніж із двома піддеревами, які називаються лівим і правим піддеревами вузла.

4- Алгоритм доступу до всіх вузлів дерева називається обходом дерева. Трьома основними способами обходу дерева є обхід зверху вниз, зліва направо та знизу вверх.

-4 Бінарне дерево є деревом бінарного пошуку, якщо для будь-якого його вузла *x* значення всіх вузлів лівого піддерева *x* не більші за значення *x,* а значення всіх вузлів правого піддерева *x* не менші за значення *x.* Така властивість називається впорядкованістю ключів у дереві. Бінарні дерева використовуються для швидкого пошуку елементів множин за заданим значенням ключа.

Контрольні запитання та завдання

1. Чим відрізняються статичні змінні від динамічних?
2. Що таке сегмент пам'яті, базис сегмента і зсунення?
3. Як розподіляється базова оперативна пам'ять під час роботи будь-якої програми, записаної на мові Разсаі?
4. Що таке динамічна пам'ять?
5. Як здійснюється доступ до статичних і динамічних змінних?
6. Означити поняття типізованого і нетипізованого покажчиків.
7. Які методи роботи з динамічною пам'яттю є в мові Разсаі?
8. В якій області пам'яті зберігаються значення покажчиків і де зберігаються дані, на які вони посилаються?
9. Дати означення лінійного списку. Яку структуру мають елементи списку?
10. Дати означення різновидів лінійного списку.
11. Які дії можна виконувати при роботі зі стеком і чергою?
12. Які покажчики потрібні для роботи зі стеком і чергою?
13. Яким чином додати та видалити перший, останній і внутрішній елементи лінійного списку?
14. Чим дерево відрізняється від лінійних списків?
15. Дати означення листка та кореня дерева.
16. У чому полягає особливість бінарних дерев?
17. Як включаються вузли у дерево?
18. Як видаляються вузли з дерева?

Вправи

1. Доповнити твердження.
   1.1. Процедури _____ використовуються для виділення динамічної пам'яті.
   1.2. Для звільнення динамічної пам'яті використовуються процедури _____
   1.3. З черги не можна видаляти _____ елементи.
   1.4. Зі стеку не можна видаляти _____ елементи.
   1.5. Степінь вузлів _____ дерева не перевищує двох.
   1.6. Вузол дерева, що не має предків, називається _____.
   1.7. Вузол дерева, що не має нащадків, називається _____.
2. Визначити результат роботи програми

```
Iуре ImoЬуIe=array[0.. Цo? ЬуЬe;
уаг питЬег :могсі;
    рІг:"ЧмоЬуЬе;
Ьеділ
  питЬег:=65535;
  рІг:=@питЬег;
  игіЬеШрЬгЧО].' '.р1;г^[1]);
епсі.
```

3. Зобразити бінарне дерево, обхід якого зліва направо дає таку послідовність значень: 49, 28, 18, 11, 19, 40, 32, 44, 83, 71, 69, 72, 97, 92, 99.
4. Записати інфіксний арифметичний вираз (6 + 2) - 5 - 8 / 4 у постфіксній та префіксній формах.

Задачі

1. Створити однозв'язний лінійний список, елементами якого є натуральні числа. Надрукувати значення елементів, розташованих між найбільшим і найменшим елементами списку.
2. Створити однозв'язний лінійний список зі слів деякого рядка, розташувавши їх у списку за алфавітом. Визначити кількість повторень кожного слова у списку. (Словом вважається обмежена пробілами послідовність символів.)
3. Файл містить текст із однаковою кількістю дужок, що відкриваються та закриваються. Побудувати чергу або стек, елементами яких є частини тексту, розташовані між парами відповідних одна одній дужок. Надрукувати номери позицій в тексті кожної пари дужок, що відкриваються та закриваються, наприклад: 8; 10; 12; 16 і т. д.
4. Створити двозв'язний лінійний список цілих чисел. Знайти елемент із введеним із клавіатури значенням. Вивести порядковий номер шуканого елемента, рахуючи з початку та з кінця списку.

5. Створити двозв'язний лінійний список цілих чисел. Ввести із клавіатури значення та порядковий номер нового елемента. Розділити список на дві частини, зробивши новий елемент останнім в одному списку і першим в іншому. Вивести значення елементів списків.
6. Реалізувати операцію додавання двох великих чисел, зображених у формі стеків. Значенням елемента стеку буде значення певної цифри числа. Під час обчислення суми її розряди також слід записувати у стек, а потім — виводити значення елементів цього стеку.
7. Створити двозв'язний лінійний список, елементами якого є слова тексту. Вивести слова, що знаходяться на парних позиціях під час перегляду списку у напрямку від голови до хвоста, та слова, розташовані на непарних позиціях під час перегляду списку у зворотному напрямку.
8. Створити однозв'язний лінійний список цілих чисел. Методом вставки упорядкувати список за зростанням і видалити із відсортованого списку всі додатні елементи. Вивести отриманий список.
9. Створити бінарне дерево цілих чисел. Визначити максимальне значення вузла дерева.
10. Створити бінарне дерево та визначити кількість вузлів на шляху від кореня до вузла, значення якого введено із клавіатури. Якщо таких вузлів декілька, вибрати будь-який із них.
11. Побудувати бінарне дерево цілих чисел, вивести його і обчислити середнє арифметичне значень усіх його вузлів.
12. Побудувати бінарне дерево і поміняти місцями найбільший і найменший його елементи. Відобразити початкове й отримане дерево.
13. Побудувати синтаксичне дерево виразу ((а + *Ь)/с)* • *ci.* Ввести значення змінних і обчислити значення дерева-формули.
14. Побудувати синтаксичне дерево виразу, що зчитується з текстового файла, вивести дерево та обчислити за ним значення виразу. Надрукувати піддерева виразу. Наприклад, для виразу із задачі 13 піддерева будуть такими: *y\ = a + Ь, y2=y1/c, y3=y2* ci.*
15. Побудувати довільне бінарне дерево та створити його копію. Вивести дерева та визначити адреси їх коренів.
16. Побудувати довільне бінарне дерево та знайти у ньому елемент із заданим значенням. Визначити рівень, на якому розташовано цей елемент.
17. Створити файл цілих чисел. Переписати компоненти цього файла у зворотному порядку, використовуючи динамічну структуру як допоміжне сховище даних.
18. Створити бінарне дерево і підрахувати кількість його листків.
19. Створити бінарне дерево цілих чисел. Видалити вузли, які містять парні числа.
20. Побудувати та вивести дерево, степінь всіх вершин якого, крім листків, дорівнює введеному натуральному числу п.

Розділ 11

Алгоритми на графах

4- Основні поняття теорії графів

4- Способи зображення графа в оперативній пам'яті

4- Задача визначення найкоротших шляхів у зваженому графі

4- Алгоритм пошуку вглибину

4- Алгоритм пошуку вшир

11.1. Поняття графа та його зображення в пам'яті комп'ютера

Граф — де сукупність непорожньої множини *V* вершин і множини *E* невпорядкованихабо впорядкованих пар вершин: *С = (У,Е), У*0,Еа Ух V.* Невпорядкована пара вершин називається *ребром,* впорядкована пара вершин — *дугою.* При цьому першу вершину з пари прийнято називати початком дуги, а другу - її кінцем. Граф, що не містить дуг, називається неорієнтованим, а граф, що містить дуги — орієнтованим або *орграфом.* Ребро (дуга) і будь-яка його (її) вершина називаються *інцидентними.* З'єднані ребром вершини називаються *суміжними.* Якщо вершина *V* є початком певної дуги, а вершина ш — її кінцем, то вершину *тю* вважають суміжною з вершиною *V,* але не навпаки.

Неорієнтований граф зображують на площині як сукупність точок, що відповідають вершинам, і як сукупність відрізків, що з'єднують точки та відповідають ребрам. Якщо граф містить дуги, вони зображуються стрілками, що з'єднують точки.

Для використання графів у програмуванні потрібно вибрати спосіб їх зображення в оперативній пам'яті. Найвідомішими структурами даних, що зображують графи, є матриці суміжності, матриці інциденцій, масиви дуг, списки суміжності. Наведемо визначення матриці суміжності та матриці інциденцій.

Нехай *п* — кількість вершин графу, *т* — кількість його ребер. Вважаємо, що всі вершини графу пронумеровані натуральними числами від 1 до *п,* а всі ребра - натуральними числами від 1 до *т. Матриця суміжності Ай*; є квадратною матрицею розмірності *п х п,* в якій

$$Adj[i,j] = \begin{cases} X \text{ вершина }] \text{ суміжна з вершиною } i \\ 0, \text{ вергшина }] \text{ не суміжна з вершиною } i \end{cases}$$

Матриця інциденцій Inc неорієнтованого графу є матрицею розмірності ге х те, в якій

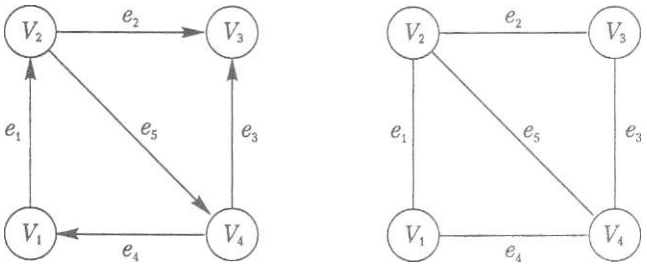
$$Inc[i,j] = \begin{cases} 1, & \text{вершина } V_i \text{ інцидентна ребру } e_j \\ 0, & \text{вершина } V_i \text{ не інцидентна ребру } e_j \end{cases}$$

Матриця інциденцій Inc оргграфу є матрицею розмірності ге х те, в якій

$$Inc[i,j] = \begin{cases} 1, & \text{вершина } V_i \text{ інцидентна дузі } e_j \text{ і є її кінцем} \\ 0, & \text{вершина } V_i \text{ і дуга } e_j \text{ не інцидентні} \\ -1, & \text{вершина } V_i \text{ інцидентна дузі } e_j \text{ і є її початком} \end{cases}$$

Приклад 11.1

На рис. **11.1** зображено орієнтований граф *О* та неорієнтований граф *С*. Нижче наведено матриці їх суміжності та інциденцій.



Я С

Рис. 11.1. Орієнтований О та неорієнтований С графи

Матриці суміжності цих графів такі:

$$A_{\text{ор}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad A_{\text{неор}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Матриці інциденцій графів *Б і Се* такими:

$$Inc\_C = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}; \qquad Inc\_p = \begin{pmatrix} -1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 1 \end{pmatrix}$$

Основним недоліком зображення графу за допомогою матриць суміжності та інциденцій є те, що ці матриці містять $0(n^2)$ елементів навіть тоді, коли граф складається з $0(n)$ вершин і $0(n)$ ребер. Більшість алгоритмів на графах потребують перегляду кожного ребра графу принаймні один раз. Тому, якщо реалізувати певний алгоритм, що має часову складність $0(n)$, з використанням матриць суміжності або інциденцій, його часова складність, як правило, зросте до $0(n^2)$.

У багатьох випадках економніший, з погляду ємнісних і часових витрат, спосіб зображення графу полягає у використанні *списку суміжності.* Кожен елемент такого списку посилається на наступний елемент, а також містить номер певної вершини і посилається на список суміжних з нею вершин. Наприклад, для орграфу Б (рис. 11.1) список суміжності має бути таким, як зображено на рис. 11.2.

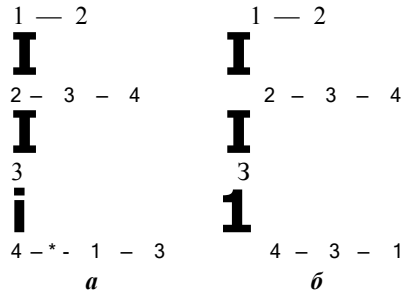


Рис. 11.2. **Список суміжності графу (а); зображення списку суміжності в текстовому файлі** *(б)*

Граф, в якому кожному ребру (і,/) відповідає число *т^,* називається *зваженим,* а число *щ* називається *вагою ребра (i,)).* Зважений граф можна зобразити матрицею вагів, що утворюється з матриці суміжності, коли її одиничні елементи замінити на відповідні ваги. Вага неіснуючого ребра вважається рівною нескінченності або нулю.

11.2, Найкоротші шляхи у графі

Шляхом, що з'єднує вершини *а* та *Ь* графу *С,* називається послідовність вигляду *вершина* 1, *ребро* 1, *вершина 2. . . . ребро N-1, вершина* де *вершина 1 = а, вершина N = Ь,* а *і-е* ребро інцидентне з і-ю та (г+1)-ю вершинами.

Для будь-яких двох вершин *і),* та графу *С* може існувати декілька шляхів, що з'єднують їх. Розглянемо алгоритм визначення у зваженому графі шляху, який має мінімальну вагу серед усіх шляхів, що починаються в заданій вершині *$1агі,* а завершуються у вершині */іпізН.* Нехай зважений орієнтований граф *С = (V, Е)* заданий матрицею суміжності *А* і ваги всіх його дуг невід'ємні. Якщо дуга від вершини О, до вершини V] відсутня, покладемо *Ау* = 00.

Одним із найвідоміших розв'язань цієї задачі є *алгоритм Дейкстри.* На кожній ітерації цього алгоритму множина *Т* вершин, найкоротші шляхи до яких від вершини *5іагі* уже визначено, збільшується на один елемент. Крім множини *Т*

розглядається масив *Раік.* Нехай P_i - множина всіх шляхів з вершини *зіагі,* останні дуги яких починаються у вершинах з множини *T,* а завершуються у вершині *і.* Значення Раїкі дорівнює довжині найкоротшого шляху з множини Р;.

Спочатку до множини *Т* включають лише одну вершину - *зїагі,* а для довільної вершини *к ф зіап* покладають $Раік_к = \infty$. Після включення до множини *Т* нової вершини *і* необхідно переглянути всі вершини *і і Т* і здійснити таке присвоєння: *Раікі* шіп*{Раікі, РаЩ + A_{ji}}.* Далі визначають величину тіп{Ра£/г,}, а вершину т, що забезпечує цей мінімум, включають до множини *Т.*

Доведемо, що елемент $Раік_m$ масиву *Раік* дорівнює довжині найкоротшого шляху від вершини *зіагі* до вершини *т.* Розглянемо довільний шлях що розпочинається у вершині *зіагі,* а завершується у згаданій вершині *т.* Нехай *к* — найперша вершина на цьому шляху, що не належить *Р.* Якщо кФт, то принцип вибору вершини *т* гарантує, що вага частини шляху ^ від вершини *зіагі* до вершини *к* є не меншою за $Раік_m$. Оскільки ваги дуг невід'ємні, то вага будь-якого шляху є не меншою за вагу його частини. Тому і вага д не менша за $Раік_m$. Якщо ж *к* збігається з *і7і,* то довжина д, згідно з побудовою масиву *Раік,* не менша за $Раік_m$.

Щоб побудувати найкоротший шлях, а не лише знайти його довжину, слід після присвоєння Раікі <~ РаЩ + А^ у певному масиві запам'ятати номер;. Алгоритм завершить свою роботу тоді, коли до множини Г буде включено вершину */іпізк.*

Оцінимо складність алгоритму. Перегляд всіх вершин я-вершинного графу, що не належать множині Г, виконується у найгіршому випадку *п* - 1 разів, і кожне його виконання потребує *0(п)* кроків: *О(п)* кроків для розрахунку довжин шляхів до кожної вершини і *0(п)* кроків для пошуку найкоротшого шляху. Отже, складність алгоритму становить $0(n^2)$.

Приклад 11.2

Реалізуємо алгоритм Дейкстри для знаходження найкоротшого шляху у графі, матриця суміжності якого записана у текстовому файлі дга1\ <М. Перше число файла визначає кількість вершин.

```
Рrogrаt ex11_1;
      {пошук найкоротших шляхів за алгоритмом Дейкстри}
C0N5I
  MaxZige = 150:     {найбільша кількість вершин графу}
  iniiniiy = 10000:  {позначення нескінченності       }
Type
  ¥eП:ex5eї = 5ei: oT 1..Max5ige:
yag
                  довжини шляхів або ваги ребер графу }
  meidïH:aggay [1..Max5ige. 1..MaxZige] oГ Inïedeg;
  T: UegïexZeї;;  {множина вершин, до яких визначено
                   найкоротші шляхи                   }
  PaIH.           {масив довжин найкоротших шляхів до
                   кожної вершини графу               }
  I/egЬex:array [1..Max5ige] oi Inïedeg;
                  {масив вершин найкоротшого шляху   }
  n: Iniedeg;     {кількість вершин графу            }
```

```
   зіагі.          {початкова вершина шляху          }
   Ii пі зіі: іпіедег; {кінцева вершина шляху          }
   1:lexl;  {файл вхідних даних, що містить кількість
              вершин графу та матрицю суміжності  }
{========== формування матриці суміжності ==============}
procesiure КeacЮaIa:
yar i.з: Iпіедег;                    {параметри циклів}
beqin
   аззідп(l, 'дгаї.ciаї'):
   гезеі(i);
   геасПп(Iп);
   lor i :=1 lo п cio
     lor ] :=i lo п cio
       КeacКImeiдШі,,]]);
   ciозе(i);
enci;
{========== виведення матриці суміжності ==============}
procesiure ОиIриЮaIa:
yar i:inIедег;                       {параметри циклів}
beqin
   мгiIe1п(Aci¡iacепсу шаігіх:');
   lor i :=1 їo п cio
   beqin
     lor ] :=1 Io п cio
       ii meiдIii[i,з]=inIiniIy IЬеп
         мгiIe('–':10)
       eize «riIe(meiдIii[i Л :10):
       мгіїеіп;
   enci;
enci;
|=============== алгоритм Дейкстри ====================}
procesiure PiпciHiпPaИ;
yar
   Miп,                          {довжина найкоротшого шляху}
   i,]; Iпіедег;                 {параметри циклів          }
beqin
{        підготовка до пошуку найкоротшого шляху.....}
   lor i:=1 lo п cio
     PaIii[i]:=inIiniIy;{найкоротший шлях не визначено}
   ;i:=5IагI;              {перша вершина найкоротшого шляху}
   T;=Ш;                   {множина T містить вершину зіагі }
   PaIb[з]:=0;      {шлях зі зіагі у зіагі маЕ довжину 0}
{                пошук найкоротшого шляху              }
«Hi1e noKIinisiї in T) cio {поки не знайдено
                    найкоротший шлях до кінцевої вершини,}
     beqin
       lor i:=1 lo n cio          {перевіряти всі вершини}
       beqin
         {якщо поточна вершина не належить множині T,
               знайти найменшу суму довжин шляхів до неї}
         il noKi in T) anci
               (PaIH[i]>PaIШ]+meiдIьи.i])
```

```
        iiieп
        Ьеділ
          {запам'ятати вагу найкоротшого шляху до
           поточної вершини та номер попередньої вершини}
          Ра1*[i]:=РаОД]+иеідіед.i]:
          VerЬex[i ]:=;];
        encs;
    encs;

{-. . . . . пошук нового елемента множини Т ---.....-}
    Мiп:=inIiniїy;  {довжину найкоротшого шляху вважати
                     нескінченною}

    ror i :=1 їo n cio
      i' noKi in T) anci {якщо вершина не належить Т. а}
        (Мiп>РаЬii[i]) {шлях до неї менший за мінімальний}
      їbeп
      Ьеділ
        Мiп:=РаїИ[i]:          {вважати шлях до поточної }
        ,]:=i:                 {вершини найкоротшим, а її
              саму - кандидатом на включення до множини Т}
      encs;
    {якщо немає шляху від початкової до поточної вершини}
      i I Мiп>=inГiniі;y їbeп
      Ьеділ
          игїїei n(' Tїїeге i 3 по мау 1тош УЄПЄХ'. зїагї,
                  ' Ьо УЄПЄХ '. Tinizії);
        reaciїn; НаЩО);          {перервати програму }
      encs
      eize T:=T+[Л;    {включити вершину до множини Т}
    encs;                        {кінець циклу miiiie }
encs:                            {кінець процедури    }
{============ виведення найкоротшого шляху ============}
procesiiire MгiЬeMiпPaЬIi;
yar i: InЬeдeг;
Ьеді n
  мгiIe1n('PaIИ:');
  i :=TinizIi;              {задати номер кінцевої вершини }
  иЬile i<> 5Ьaгi: cio      {поки не досягнуто номера      }
  Ьеділ                     {початкової вершини            }
    мгїe(i.'<-'):           {виводити номери вершин шляху  }
    i:=Уегїех[i]:           {перейти до попередньої вершини }
  encs;
  мгїeInCзЬaгЬ);            {вивести номер першої вершини  }
  мгiЬeInC'I_enдIH=.РаїИITinizИ]): (вивести довжину шляху}
encs;
|=============== основний блок програми ================}
Ьеділ
  KeacЮaIa;                 {ввести дані з файла           }
  ОиїриЮaia;                {вивести матрицю суміжності    }
  repeai                    {ввести номер початкової вершини}
    мгiIe1n('Eniег Бїагї yeгЬex <=', п);
    reaci' n(зiагї);
```

```
  иnііі зіаг1:<=п;
  гереаі:                 {ввести номер кінцевої вершини }
    мгі£еln(Епіег їіпізН УСПЕХ <=', п);
    геасі п('Ті пі з Ь);
  иnііі їіпізИ<=п;
  РіпсіМіпРаїН;           {застосувати алгоритм Дейкстри }
  МгїеНіпРаїт;            {вивести  результати           }
  геасі п;
enci.
```

```
і • C:\BP\BIГЧ\EX11 _1 .EXE
 Ас^асепсу шаігіх:
                                                       -        -   і
               7        2        1        5        -
                                 5                 -
       -       3        -                 3        Z
                                                   5        2
               8                          1        -        6

 Епіег 5Іагі чегіех <=7
 А
 IIЕпіег ГЇПІ5П иегіех <=7
 7
 РаІП:
 ;7<-5<-4<-2<-3<-1
 1еnдІП=11
 \ш                                                         2Г1
```

Рис. 11.3. **Результати роботи програми ex11_1.**
Пошук найкоротшого шляху в орієнтованому графі

11.3. Обхід графу

Численна кількість алгоритмів на графах ґрунтується на переборі всіх або частини їх вершин. Такий перебір називається обходом графу. Найвідоміші алгоритми обходу графу - це пошук вглибину та вшир.

11.3.1. Пошук вглибину

Розглянемо алгоритм *пошуку вглибину* у зв'язному неорієнтованому графі. Пошук починається із заданої стартової вершини і триває доти, доки не буде знайдено певної кінцевої вершини або поки є вершини, що утворюють маршрут пошуку. Стартову вершину пошуку позначимо літерою в. Виберемо довільне ребро (*V*, *u),* інцидентне *V.* Якщо вершина *u* вже відвідана, вибираємо інше ребро, що є інцидентним *V.* У протилежному випадку вершину *u* вважаємо стартовою і повторюємо процедуру пошуку. Якщо пошук з вершини *u* завершується безуспішно, повертаємося до вершини *V* і продовжуємо пошук ребер, інцидентних вершині *V* і таких, що раніше не відвідувалися. Вибір нових ребер триває доти, доки не вичерпаються всі ребра, що є інцидентними *V,* або доки пошук кінцевої вершини не завершиться успішно. Зауважимо, що під час пошуку вглибину в орієнтованому графі слід переглядати лише ті дуги, що виходять зі стартової вершини.

Важлива властивість алгоритму пошуку вглибину полягає в тому, що моменти виявлення вершини *V* і завершення процесу обробки списку суміжних з нею вершин утворюють правильну дужкову структуру [11]. Якщо позначити факт виявлення вершини V дужкою, що відкривається, а завершення процесу обробки суміжних з нею вершин - дужкою, що закривається, то послідовність дій утворить правильно побудований вираз із дужок.

Оцінимо часову складність алгоритму, припускаючи, що граф має *n* вершин і т ребер. Пошук верпіини, що не була відвідана, виконується не більше ніж за *0{n)* кроків, а кількість кроків, що їх необхідно здійснити для перебору всіх інцидентних певній вершині ребер, становить не більше ніж *0(m).* Отже, часова складність пошуку вглибину становить *0(n + m*), де *n* — кількість вершин графу, *m* - кількість його ребер.

Приклад 11.3

Використаємо алгоритм пошуку вглибину для знаходження шляхів від стартової вершини до кінцевої в неорієнтованому графі. Структура графу записана в текстовому файлі, перший рядок якого містить кількість вершин графу, а кожний наступний рядок — два числа, що є номерами певних суміжних вершин. Тобто кожен рядок файла, починаючи з другого, визначає ребро графу. Номери стартової та кінцевої вершин користувач вводить з клавіатури.

Для зображення графу використаємо список суміжності. Тип вершини назвемо ТУеr. Запис типу ТУеr міститиме такі поля: поточний номер вершини, кількість суміжних із нею вершин, масив покажчиків на суміжні вершини, ознаку відвідування вершин. Сам граф задамо масивом a , елементи якого будуть покажчиками на записи ТУеr.

Ініціалізація масиву вершин даними, що зчитані з вхідного файла, виконується у процедурі І пі "Ь, з якої викликається процедура зв'язування суміжних вершин ИпкУегТех. Зазначимо, що, оскільки розглядається неорієнтований граф, одному ребру відповідають два зв'язки між вершинами.

Власне пошук вглибину здійснює рекурсивна процедура зеагсЬ, що є локальною процедурою щодо процедури ОерТіїРігзТЗеагсЬ (пошук вглибину). У процедурі ОерТЬРігзТЗеагсЬ запрограмовані введення початкової і кінцевої вершин для пошуку, помітка початкової вершини як такої, що вже опрацьована, і всіх інших вершин як таких, що потребують перегляду, а також виклик процедури 5еагсЬ.

```
рrодrат  е х 11_2;              {пошук  вглибину                  }
и$€5 сгТ;
Type
  РТгУег » ^ТУеr;              {тип покажчика на вершину графу }
  ТУеr = recorci               {тип  вершини  графу              }
    уегТех:аггау[1..50] оТ РТгУег; {масив суміжних вершин}
    к.                         {кількість суміжних вершин     }
    питЬег:іпТедег;            {номер  вершини                   }
    тагк: Ьооїеап;             {ознака відвідування вершини   }
  enci;
yar а:array[1..50] оТ РТгУег;    {масив покажчиків
                                    на вершини графу     }
```

```
    n: integer;          {кількість вершин графу
    T:TextT;             {текстовий файл зі списком суміжності
    key:cChar;           {символ обраної користувачем дії
    startT,              {номер початкової вершини обходу графу
    Ti pi zii: integer; {номер кінцевої вершини
{======== встановлення зв'язків між вершинами графу ========
procedure InitLink(var V, u: PTgVer);
            {v,u - покажчики на вершини, що зв'язуються}
begin
                          {включити u В СПИСОК суміжності V}
  inc(v^.k); v^.Edge[v^.k]:=u;
                          {включити V у список суміжності u}
  inc(u^.k); u^.Edge[u^.k]:=v;
end;
|============== ініціалізація графу з файла ================
procedure InitT;
var i,                          {параметр циклу
    iV,iu:integer;              {номери суміжних вершин
    Ver:PTgVer;                 {покажчик на вершину графу
begin
  reset(T);                     {відкрити файл списку суміжних вершин
  readln(T,n);                  {зчитати кількість вершин графу
  for i:=1 to n do              {створити масив покажчиків на
  begin                         (вершини графу
  new(Ver);                     {виділити пам'ять для покажчика
    Ver^.nitNer:=i;             {визначити номер вершини
    Ver^.k:=0;                  {задати КІЛЬКІСТЬ суміжних вершин
    a[i]:=Ver;                  {записати покажчик на вершину в масив
  end;
  while not eof(T) do           {поки не досягнуто кінця файла
  begin
    readln(T, iV, ch);          {читати номери суміжних вершин
    InitLink(a[iv], a[iu]);     {зв'язати вершини
  end;
  close(T);                     {закрити файл
end;
{=====-============== пошук вглибину ======================
procedure OerTiRigzTZeargcL;
var
  m:array [1..50] of integer; {масив номерів вершин маршруту
  i,,] :integer;              {лічильники циклів
{================ алгоритм пошуку вглибину =================
procedure 5eargcI(V:PTgVer);
                        {V - покажчик на початкову вершину
var i:integer;
begin
  if V^.nitNer = Tinizii then   {якщо досягнуто кінцевої
  begin                         {вершини,
    mgiTeC PaTi: >> ');         {вивести маршрут
    for i :=1 to ,1 do
      mgiTe(m[i], ' -> '):
    mgiTeln;
```

```
    exii;
  encl
  eize                    {якщо кінцевої вершини не досягнуто,}
    Tor i:=1 To Va, к cio  {перегляд усіх суміжних вершин}
    Ьедіп
                           {якщо поточна вершина не помічена,}
      iT noT VA.УЄГЇЄХ[I.тагк Tiien
      Ьедіп
        у^.уегТех[i]^.тагк:=Тгие;   {ПОМІТИТИ вершину }
        ,I:=д+1;  {збільшити лічильник помічених вершин}
        «[,]]:=УA.УЄГТЄХ[I]^.пишЬег; {запам'ятати номер
                                      вершини          }
        5еагсЬ(у^.УегТех[i]);    {перейти до наступних}
                                 {вершин              }
        у^.уегТех[i]^.тагк:=Та1зе; {зняти помітку з   }
                                 {пройденої вершини   }
      encl:
    encl;
encl;
                  {основний блок процедури пошуку вглибину}
Ьедіп
  сігзсг;
  мгіТе1п('«« ciepTH TігзТ зеагсН »»');
  мгіТе1п('**************************·)·
  игіТе(iniTiaI уегТех :'); reacl п(зіагі);
  мгіТе('Тегтіпаї уегїех :'); reacl п(Ті пі зН):
  Tor i:=1 To п cio   {усі вершини вважати непоміченими}
    a[i]^.тагк:=Та1зе;
  a[зїагТ]^.тагк:=Тгие;     {помітити початкову вершину}
            {задати номер початкової вершини у маршруті}
  «[,]] :=зТагТ:
  ЗеагсИ(a[зТагТ]); {виклик рекурсивної процедури пошуку}
  геасПп;
encl;
{                основний блок програми ================}
Ьедіп
  аззідп(Т,'дгарЬ.Iin');
  IniT;                              {створити граф  }
  repeaT                             {нескінченний цикл}
    сігзсг;
    мгіТеїпСМитЬег оТ уегТехез =', п);

    мгіТе1п(ргезз <ЕпТег> То ЗОІУЄ');
    мгіТе!п(<Езс> - То exiT');
    кеу:=геасікеу;
    сазе кеу оТ
      #13 : ОерТИРігзТЗеагсЬ; {натиснуто клавішу ЕпТег }
      #27 : Иаїї;              {натиснуто клавішу Езс  }
    encl;
  илТіі Таїзе;
encl.
```

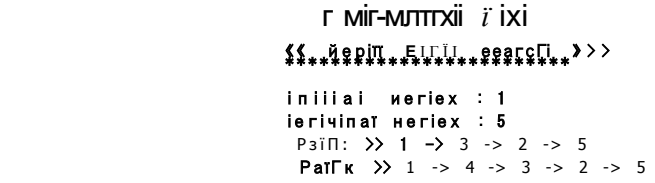


Рис. 11.4. Результати роботи програми ехі 1_2. Пошук вглибину

Зображені на рис. 11.4 результати отримані з такого вхідного файла:

11.3.2. Пошук ушир

За алгоритмом *пошуку вшир* здійснюється обхід вершин графу в порядку збільшення відстані від стартової вершини. Відстань між вершинами *а* і *Ь* вважається рівною кількості ребер на найкоротшому шляху від *а* до *Ь*. Отже, спочатку переглядається стартова вершина, потім - суміжні з нею вершини, що входять до списку перегляду. Кожна ітерація алгоритму полягає у послідовному виборі всіх вершин зі списку перегляду і додаванні до цього списку вершин, що є суміжними з вибраними. При цьому вершини не можуть переглядатися двічі, а тому під час перегляду вершину слід помічати. Помічені вершини надалі не розглядаються.

Список перегляду найзручніше зображувати у вигляді черги. При перегляді вершини всі суміжні з нею непомічені вершини додаються до черги, а сама вершина з черги вилучається. Пошук виконується доти, доки не буде знайдено кінцевої вершини або доки список перегляду не стане порожнім.

Часова складність пошуку вшир, так само як і пошуку вглибину, становить *0(n + m)*, де *n* - кількість вершин графу, *m* — кількість його ребер. Справді, кожна вершина додається до черги і вилучається з неї не більше ніж один раз, а кількість ітерацій циклу пошуку суміжних вершин з точністю до сталого множника дорівнює кількості ребер графу.

Приклад 11.4_

Використаємо алгоритм пошуку вшир для знаходження шляху від стартової вершини до кінцевої в неорієнтованому графі. Інформація про граф записана у текстовому файлі, який має ту саму структуру, що і файл вхідних даних для програми пошуку вглибину.

Процедури Іпіі і Ипкі'егіех у тексті програми не наводимо, оскільки вони цілковито збігаються з відповідними процедурами програми ех11_2. Пошук ушир вер-

шини TiпizH, що розпочинається з вершини ziагі, виконує процедура BreacШi-PirgIZearcIї («пошук ушир»). Усі вершини, що є суміжними з поточною і не були помічені, додаються до черги за допомогою процедури AcMИ zi. Черга переглядається, починаючи від вершини PCиг, тому виконання оператора PCиг: =PCиг^A. Mexi є дією, логічно еквівалентною видаленню вершини з черги. У разі знаходження кінцевої вершини або вичерпування всіх можливостей пошуку, чергу слід очистити за допомогою процедури Ciear. Нарешті, рекурсивна процедура Оиіриі здійснює виведення знайденого шляху. Для посилання на попередню вершину шляху використовується покажчик **PГЄУ**.

```
prodgat ex11_3:        {пошук ушир                          }
izez cгi;
iype
  PIгУег = ^ТУег;   {тип покажчика на вершину графу  }
  ТУег = recorci      {тип вершини графу                 }
    уегIех:аггау[1..50]joT Pii-Уег:{масив суміжних вершин}
    к,                {кількість суміжних вершин       }
    питЬег:іпіедег;   {номер вершини                   }
    тагк: Ьооїеап;    {ознака відвідування вершини     }
  epci;
  PEЇет = ^TE1et;   {тип покажчика на елемент черги  }
  TEIet = recorci     {тип елемента черги              }
    РУег: PIгУег;     {покажчик на поточну вершину     }
    PPгош : PEIeш;    {покажчик на попередню вершину   }
    PГЄУ, 4           {покажчик на попередній елемент черги }
    MexT:PEIeш;       {покажчик на наступний елемент черги }
  ЄПС;
уаг PBeд,PEпсі    {покажчики на початок і кінець черги }
    PCиг:PEЇет;   {покажчик на поточний елемент черги  }
    I:IexT;       {текстовий файл зі списком суміжності }
    кеу:сііаг;    {символ вибраної користувачем дії    }
    ziагі,        {номер початкової вершини обходу графу}
    Ті piзЬ:іпТедег; {номер кінцевої вершини обходу   }
    a:array[1..50] ci PTгУег; {масив покажчиків на
                                 вершини графу       }
    п: іпТедег;               {кількість вершин графу }

{== процедуру ИпкУегТех наведено у програмі ex11_2 ===}
{===== процедуру IniT наведено у програмі ex11_2 ======}

{=========== додавання елемента до черги ==============}
procesiиre ACICHIЗKУ: PTгУег; Я:PE1et);
{V - покажчик на вершину, що додається до черги:
 Я - покажчик на попередній елемент черги}
уаг р: PEЇеш:       {поточний покажчик                 }
beдiп
  пем(р);           {виділити пам'ять для елемента черги}
  р^.РУег:=у;       {покажчик на поточну вершину       }
  р.PPгот:=ч;       {покажчик на попередню суміжну
                       вершину                         }
  р^.Ргеу=PEпсГ.Ргеу;{зв'язати поточний елемент      }
  р.№.хI:=PEпсі;      (із кінцевим у черзі             }
```

```
  PEпсГ Ргеу^.№.хТ:=р;
  PEпсІ^.Ргеу:=р;
епсі:
1============ видалення елемента з черги ==============}
procedure Oe1i_izT(e:pE1eт);
        {параметр - покажчик на елемент, що видаляється}
  var p: PEIeт;
begin
  if e = nil then exiT:  {якщо елемент порожній,
                          вийти з програми          }
  e^.MexT^.РГЄУ:=e^.РГЄУ; {переадресувати покажчики  }
  e^.РГЄУ^.№.хТ:=e^.№.хТ;
  ciizrozeCe);           {звільнити пам'ять з-під елемента}
end;
{========== виведення поточного елемента черги ========}
procedure OиїриТ(e:pE1eт);
begin
    if e = nil then exii; {якщо елемент порожній.
                           вийти з програми          }
    оиТриТ(e^.РРгот);     {вивести решту черги      }
    if e.РУег^.пишЬег ◇ TipizH then {елемент не     }
      мгіТе(e.Р¥ег^.питЬег, ' -> ') {останній       }
    else игіТе(e.Р¥ег^.питЬег. ' '); {елемент останній}
end;
{================= видалення черги ====================}
procedure Cieaг;
var PCиггепТ, PPe1:PE1eш;
begin
  PCиггепТ:=PBед^.№.хТ;
  miiiie PCиггепТ ◇ PEпсі do {ПОКИ не досягнуто      }
  begin                      {кінця черги            }
    POeI:=PCиггепТ; {покажчик на елемент, що видаляється}
    PCиггепТ:=PCиггепТ^.№.хТ:{переадресувати покажчик
                              на наступний елемент    }
    OєИізТ(POeI);            {видалити елемент       }
  end;
end;
I====================== пошук ушир ====================}
procedure BгeaciTЬPigzTZeагсіі;
var i:inTедег; {параметр циклу}
begin
  сігзсг:
  УУГІЇЄІП('«« ЬгeaciTИ TigzT zeагсіі »»');
  мгі Теї п('**********************');
  мгіїе('iniTia! УЄГТЄХ : '); reacii п(зіагї);
  мгіТе(Тегтіпаї УЄГТЄХ : '); reacii п(Ті пі зіі);
  AciciLi5T(a[5TarT]. пії); {додати в чергу вершину  }
  for i:=1 to n do         {задати ознаки відвідування }
    a[i]^.шагк:=Ta1ze;     {вершин графу             }
  PCиг:=PBед^.MexT:        {вибрати початок черги    }
  while PCиг ◇ PEпсі do {поки не досягнуто кінця черги}
  begin
    PCиг^.РУег^.шагк:=Тгие; {відмітити вершину графу }
```

```
    if PCur^.Vert^.visited = true then
    begin                    {якщо вершина остання в черзі, то}
      writeln('Path : ');
      outputQueue(PCur);                    {вивести чергу}
      clear;                                {очистити чергу}
      readln;
      exit;                                 {вийти з циклу}
    end;
                             {якщо вершина графу не остання}
    for i:=1 to PCur^.VCount do
    begin                    {якщо вершина не відмічена}
      if not PCur^.Vert^.edges[i]^.mark then
                             {додати вершину до черги }
        AddToQueue(PCur^.Vert^.edges[i], PCur);
    end;
    PCur:=PCur^.Next;        {перейти до наступної вершини}
  end;
  writeln('>> Path not found <<');
  readln;
  clear;                                    {очистити чергу}
end;
                основний блок програми
begin
  assign(f,'graph.in');
  mem[PVer]; mem[PEnd];
  PVer^.Vert:=nil; PVer^.No:=1; PEnd:=PVer;
  PEnd^.Vert:=PVer; PEnd^.No:=nil;
  init;
  repeat
    setcur;
    writeln('Current vertex = ', n);

    writeln('press <Enter> to continue');
    writeln('<Esc> - to exit');
    key:=readkey;
    case key of
      #13 : BreadthFirstSearch;{натиснуто клавішу Enter}
      #27 : halt;              {натиснуто клавішу Esc }
    end;
  until false;
end.
```

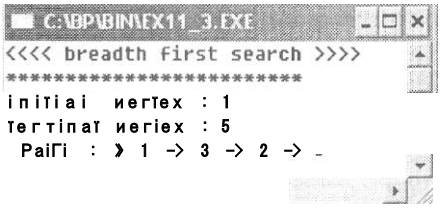


Рис. 11.5. **Результати роботи програми ex11_3. Пошук ушир**

Зображені на рис. 11.5 результати отримані з такого файла вхідних даних:

```
5
1 3
1 4
2 3
2 5
3 4
```

Висновки

4- Граф — це сукупність непорожньої множини У вершин і множини *Е* невпорядкованих або впорядкованих пар вершин: *С = (V, Е), V Ф* 0, *Е* с Ух У.

4- Невпорядкована пара вершин називається ребром, впорядкована - дугою. Граф, який містить тільки ребра, називається неорієнтованим. Граф, що містить дуги, називається орієнтованим або орграфом.

4- Ребро (дуга) і будь-яка його (її) вершина називаються інцидентними. З'єднані ребром вершини називаються суміжними. Якщо вершина V є початком дуги, а вершина *m* — її кінцем, то вершина ж є суміжною з вершиною *V,* але не навпаки.

4- Найвідомішими структурами даних, що зображують графи в оперативній пам'яті, є матриці суміжності, матриці інциденцій, масиви дуг, а також списки суміжності.

4- Граф, у якому кожному ребру (і,]) відповідає число Шу, називається зваженим, а саме число *му* — вагою ребра *(i,].*

4- Для пошуку найкоротших шляхів між парами вершин зваженого графу з невід'ємними вагами дуг застосовується алгоритм Дейкстри. Послідовність кроків є такою: відмічаємо стартову вершину; для кожної невідміченої вершини будуємо шляхи, що з'єднують її з останньою відміченою вершиною; вибираємо з цих шляхів найкоротший і відмічаємо наступну вершину. Процес триває доти, доки залишаються невідмічені вершини.

4- В основу багатьох алгоритмів на графах покладено систематичний перебір їх вершин. Такий перебір називається обходом графу.

4- Пошук вглибину в графі починається із заданої стартової вершини і триває доти, доки не буде знайдено певної кінцевої вершини або доки є вершини, що утворюють маршрут пошуку.

4- Алгоритм пошуку вшир здійснює обхід вершин графу в порядку збільшення відстані від стартової вершини. Відстань між двома вершинами вважається рівною кількості ребер на найкоротшому шляху, що з'єднує вершини.

Контрольні запитання та завдання

1. Дати визначення графу.
2. Як зображується граф в оперативній пам'яті?

3. Для яких задач кожен із способів зображення графу є найбільш ефективним?
4. Що таке матриця суміжності і матриця інциденцій?
5. Викладіть суть алгоритму Дейкстри.
6. Що таке обхід графу і для яких цілей він застосовується?
7. Поясніть, у чому полягає відмінність між алгоритмами обходу графу вглибину та вшир.
8. Якою є часова складність алгоритмів обходу графу вглибину та вшир?
9. Наведіть приклади задач, для розв'язання яких використовуються алгоритми обходу графу.

Задачі

1. Перевірити, чи є заданий неорієнтований граф зв'язним.
2. Циклом у графі називається маршрут, початкова і кінцева вершини якого збігаються. Перевірити, чи містить заданий неорієнтований граф хоча б один цикл.
3. Задано неорієнтований граф. Застосувавши алгоритм пошуку ушир, визначити всі вершини графу, відстань яких від заданої вершини 5 становить *<1.*
4. Існує ./Уміст. Для кожної пари міст (і,і) можна побудувати шлях, який з'єднає їх та не буде заходити до інших міст. Вартість будівництва такого шляху становить йу. Визначити найдешевший спосіб будівництва шляхів, іцо дозволив би потрапити з кожного міста до будь-якого іншого.
5. Знайти найкоротший маршрут, що розпочинається і завершується в заданій вершині орієнтованого графу, проходячи через всі його вершини (якщо такий маршрут існує).
6. Заданий орієнтований граф з *N* вершинами. Обчислити кількість різних шляхів між усіма парами вершин графу.
7. Карта радіоактивного забруднення місцевості є прямокутною таблицею A^x M, у клітинках якої записані дані про рівень забруднення відповідної ділянки. Знайти шлях із лівої верхньої клітинки таблиці до правої нижньої, сумарна доза радіації на якому є найменшою. Ділянки шляху паралельні межам таблиці.
8. Застосувавши алгоритм пошуку вглибину, розробити програму пошуку в неорієнтованому зв'язаному графі шляху, який проходить один раз через кожне ребро в кожному напрямку.
9. Локальна мережа містить A^r комп'ютерів, окремі з яких заражені вірусом. Кожен канал зв'язку з'єднує певні два комп'ютери. Мережа вважається повністю ураженою, якщо жоден незаражений комп'ютер не з'єднаний із незараженим. Визначити мінімальну кількість комп'ютерів, зараження яких призведе до повного ураження мережі.
10. Певний механізм складається з деталей, які, у свою чергу, містять інші деталі. На виробництво кожної з них або на її складання з інших деталей витрачається певний час. Визначити загальний час, необхідний для виготовлення механізму, зобразивши послідовність складання деталей у вигляді орієнтованого графу.

Задачі підвищеної складності

Нижче наведено умови 15 задач з програмування, що пропонувалися на районних, міських та всеукраїнських олімпіадах з інформатики протягом декількох останніх років. Усі задачі вимагають від студента не лише застосування нетривіальних прийомів програмування, але й навиків нестандартного логічного мислення, а також загальної ерудиції.

1. Куб

У декартовій системі координат розташовано куб, координати вершин котрого дорівнюють невід'ємним цілим числам, що не перевищують 15. Одна з вершин куба збігається з початком координат. Скласти програму CIIBE, яка обчислювала б квадрат відстані на поверхні куба між двома точками, заданими координатами. (Відстанню між двома точками на поверхні куба називається найменша довжина ламаної, що з'єднує дві задані точки і повністю лежить на поверхні куба.)

Вхідний файл **cube.dat** містить такі дані. У першому рядку записано довжину сторони куба, у двох наступних — по 3 натуральних числа, що задають координати *x, y, z* двох точок на поверхні куба.

Вихідний файл cube.res створюється із вхідного файла cube.dat дописуванням до нього нового рядка, що містить квадрат довжини ламаної, яка з'єднує дві точки.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

cube.dat
```
10 2
1  2  0
2  1  0
```

cube.res
```
10 2
1  2  0
2  1  0
2
```

2. Прогноз

Задано *m* прогнозів щодо послідовності фінішування *m-n* спортсменів (тобто щодо того, хто з них яке місце займе у змаганні). У кожному прогнозі вгадано лише *n* місць, причому кожне місце вгадано лише в одному прогнозі. Створити програму РОК.ECA5T, що визначає порядок фінішування спортсменів.

Перший рядок вхідного файла Torecasi.dat: містить два натуральних числа — *m* та *n*. Наступні *mn* рядків містять прогнози, в яких кожного спортсмена позначено однією латинською буквою.

Рядки вихідного файла rorecasi.res міститимуть букви, що позначають спортсменів у тому порядку, в якому вони начебто мають фінішувати (у кожному рядку по одному можливому варіанту, у різних рядках - різні варіанти).

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

Torecari.ciai
2 2
acbcI
cibca

Torecasi rez
abcci
cicba

3. Автобус

У місті декілька кільцевих автобусних маршрутів, на кожному з яких курсує один автобус. Деякі маршрути мають спільні зупинки. Коли два або більше автобусів зустрічаються на зупинці, водії обмінюються новинами. Рух вони розпочинають одночасно, і кожний знає одну новину, якої не знає інший водій. Час проїзду від однієї зупинки до іншої однаковий для всіх автобусів. Скласти програму B115, яка дала б змогу визначити, чи може кожен водій дізнатися про всі новини від своїх колег, якщо час його роботи необмежений

Перший рядок вхідного файла Ьиз.ciai містить натуральне число *ci* - кількість автобусів (маршрутів). Наступні *ci* рядків описують окремі маршрути: перше число у рядку визначає загальну кількість зупинок на даному маршруті, наступні *к* чисел - послідовність їх номерів (нумерація зупинок є спільною для всіх маршрутів). Рух на кожному маршруті починається із зупинки, номер якої вказується першим.

У вихідному файлі Ьи5.rez буде зберігатися число 1, якщо кожен водій може дізнатися про всі новини, та 0 - у протилежному випадку.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

Ьиз.ciai:
1 3
3 1 2 3

Ьиз.re5
1

4. Квадрат

На площині задано координати вершин трикутника: $(X_b Y_x)$, (X_2, Y_2), Скласти програму 5£ШАКЕ, яка за вказаними координатами знаходить довжину І сторони квадрата мінімальної площі, в який можна вписати цей трикутник (всі вершини трикутника мають знаходитися або всередині квадрата, або на його сторонах). Врахувати такі обмеження: координати вершин трикутника мають бути дійсними числами в діапазоні від 0 до 10 000; його довжину необхідно визначити з точністю до 10^{-4}.

Вхідний файл зяиаге.сЗаї містить один рядок з координатами вершин трикутника, поданими у такій послідовності: $X^{\wedge}$ Y_x X_2 Y_2 X_3 Y_3.

У вихідний файл зяиаге.геБ необхідно записати одне число — значення I, тобто довжину сторони шуканого квадрата.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

5яиаге.ciai
0.0 0.0 1.1 0.0 0.0 1.1

зяиаге.гез
1.1

5. Лабіринт

Щоб дістатися джерела живої води, мандрівнику необхідно пройти через лабіринт. Використовуючи магію, мандрівник може проходити крізь стіни. Лабіринтом є складений з NxN клітин квадрат, на межах між деякими клітинами якого побудовані стіни. В кожний момент часу мандрівник може знаходитися в одній і тільки в одній клітині лабіринту. Одним ходом вважається переміщення мандрівника в сусідню по горизонталі чи по вертикалі клітину. Мандрівник може проходити крізь стіни *К* разів. Написати програму МА2Е, що визначатиме мінімальну кількість ходів, за яку мандрівник може дістатися джерела в клітині з координатами *(P,0),* починаючи шлях в клітині з координатами (1,1).

Вхідний текстовий файл таге.ciai: в першому рядку містить числа A^r, *К, P,* (2. В кожному з наступних IУ- 1 рядків записано *N* цілих чисел, які є ознаками існування горизонтальних стін між клітинами. Останні *N* рядків містять по N - 1 цілих чисел - ознак існування вертикальних стін між клітинами. Відсутність стіни позначається числом 0, її наявність - числом 1 ($2 < N < 200$, $1 < K < 250$, *1 <X, Г<А).*

В єдиному рядку вихідного текстового файла таге.гез має зберігатися мінімально можлива кількість ходів.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

таїе.ciai:
3 1 2 3
0 0 0
0 1 о
1 о
1 о
о о

таге.гез
З

6. Шифр

Деяке повідомлення шифрувалося таким чином: кожне слово вихідного тексту перекладалося за допомогою словника на спеціальну таємну мову, потім із повідомлення вилучалися знаки пунктуації та пробіли між словами, тобто отримали суцільний рядок символів. Згодом у цей рядок у довільному порядку вставлено декілька символів. Зашифрована фраза таємної мови є послідовністю символів а..*г* латинського алфавіту. Написати програму С1РНЕК, яка за заданим закодованим

рядком і словником таємних слів визначає найменшу кількість символів, котру потрібно викреслити із рядка для того, щоб рядок, який залишився, можна було подати у вигляді послідовності слів словника. Слова можуть повторюватись. Вважається, що порожній рядок можна подати за допомогою слів будь-якого словника.

У першому рядку вхідного файла **cipHer.ciaï** містяться два цілих числа: *N* - довжина зашифрованого рядка (1 < *N* < 100), *M* - кількість слів у словнику (0 < *M* < 100). Сам зашифрований рядок знаходиться у другому рядку файла. В кожний з наступних *M* рядків записане слово з словника, довжина якого становить не менше 1 та не більше *N*.

В єдиному рядку вихідного файла **cipber.rez** має бути записане натуральне число, що визначає мінімальну кількість викреслювань, після яких зашифрований рядок можна подати у вигляді послідовності слів словника.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

cipber.ciaï
11 5
aЬaicitЬcizya
aЬa
a
Ьacїї
CIZy
2ЄГ0

cipHer.rez
2

7. Лінії зв'язку

Для забезпечення доступу до мережі Інтернет всіх шкіл міста потрібно провести лінію зв'язку від міського провайдера до однієї зі шкіл, а також між деякими школами. Відома вартість встановлення ліній зв'язку між окремими парами шкіл. Написати програму 5CHOOID вхідними даними якої є величини вартості проведення ліній зв'язку між деякими (не обов'язково всіма) парами шкіл. Програма має визначати вартості двох найдешевших схем забезпечення шкіл доступом до мережі.

Перший рядок вхідного файла **БcHooїз.ciaï** містить два натуральних числа: *N* - кількість шкіл у місті (3<IУ< 100), *M* - кількість можливих ліній зв'язку між ними. В кожному з наступних *M* рядків записано по три числа, A_b $B_{,,}$ **C„** де А„ $B_{г,}$ - номери шкіл, *Ci* — вартість проведення лінії зв'язку від школи *A,* до школи 5;. Школи пронумеровані числами від 1 до *N*. Вартість лінії зв'язку $C_г$ не може перевищувати 300.

В єдиному рядку вихідного файла **scbooїз.rez** мають міститися два знайдених натуральних числа — та 5_2 і 5_2). Ці числа можуть збігатися, якщо існує більше одного варіанта схеми прокладки мережі з найменшою вартістю.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

scHooїз.ciaï:
5 8
1 3 75

```
З 4 51
2 4 19
3 2 95
2 5 42
5 4 31
1 2 9
3 5 66
```

```
5CH00I5.ГЄ5
110 121
```

8. Калькулятор

Задано алгебричний вираз, котрий містить дійсні числа і знаки арифметичних операцій +,-,*. Потрібно написати програму ЕХРКЕ53ЮМ, яка розставить дужки в цьому виразі так, щоб його значення стало максимально можливим.

Вхідний файл ехргеззіоп.сіаї містить вираз довжиною не більше ніж 250 символів. Вираз складається не більше ніж з 50 чисел, кожне з яких лежить у діапазоні від 0 до 100. Пробіли всередині чисел не допускаються.

У перший рядок вихідного файла ехргеззіоп.гез необхідно записати максимально можливе значення виразу, отриманого після розстановки дужок, а в другий рядок — сам цей вираз. Якщо варіантів декілька, потрібно вивести їх усі.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

ехргеззіоп.сіаі:
```
1+2-3.0*4
```

```
ЄХрГЄ53І0ПДГЄ5
0
((1+2)-3)*4
```

9. Алхімія

Відомі *К* видів речовин і Л[г] типів хімічних реакцій. У результаті кожної реакції з декількох вхідних речовин утворюється декілька вихідних. Проведення реакції вимагає фіксованого часу. Речовини, отримані в результаті реакцій, можна виділяти в *чистому* вигляді для подальшого застосування — вони утворюються в достатній кількості. Речовину, отриману в результаті реакції, що завершилася, дозволяється відразу ж використовувати в інших реакціях. Реакції можуть відбуватися одночасно. Написати програму АІХНЕМУ, що за інформацією про речовини і тип реакції визначає, за який найменший час можна одержати якусь певну речовину.

Перший рядок вхідного файла аІсНету.сІаі: **містить** чотири цілих числа: *К* - кількість речовин (З <*К*< 250), ЛГ- кількість реакцій (З <*М*< 500), *М*— кількість наявних напочатку речовин та номер цільової речовини (1 < *М* < *К).* Далі слідують ЛГ блоків, що описують реакції. Кожний блок складається з трьох рядків: перший містить натуральне число - час, потрібний для проведення реакції, другий - кількість речовин, що вступають у реакцію, і їх перелік, третій - кількість речовин, що утворюються в результаті реакції, і їхній перелік. Речовини, наявні на початок проведення досліду, пронумеровані числами від 1 до *М,* а усі інші - числами від *М*+ 1 до *К.* Сумарний час проведення всіх реакцій не перевищує 2-Ю^9.

Єдиний рядок вихідного файла **aiciietv.rez** повинен містити ціле число - визначений мінімальний час, за який може бути отримана цільова речовина. Якщо таку речовину одержати неможливо, рядок міститиме число -1.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

aicHetv.cial:

```
4 3  1 4
8
1 1
1 4
3
1 1
2 2 3
2
2 1 3
1 4
```

aiciietv.rez

```
5
```

10. Багатокутники

На площині задана множина з ЛГ багатокутників, жодні два з яких не мають спільних точок. Для кожного г'-го багатокутника існує *P,* багатокутників, усередині яких він міститься, і ЛГ - 1 - Р[багатокутників, що містяться усередині нього ($0 < P_i < N$ - 1). Написати програму РОЬУССЖЗ, яка для кожного багатокутника визначатиме кількість багатокутників, усередині яких він розташований (багатокутники задані координатами своїх вершин).

Перший рядок вхідного файла **poiydopiz.ciai** містить ціле число *N* — кількість багатокутників (3 < ЛГ< 10000). Наступні ЛГ рядків файла описують Л[багатокутників: *(i* + 1)-й рядок файла описує *i*-й багатокутник. Перше ціле число *Ci* визначає кількість вершин багатокутника (3 < C_i < 20), наступні С, пар дійсних чисел - **ко**ординати вершин багатокутника в порядку їх обходу. (Координатами вершин є цілі числа, що належать діапазону від -2 000 000 000 до 2 000 000 000.)

Єдиний рядок вихідного файла poïydopiz**.rez** повинен містити ЛГ чисел; *i-е* число дорівнює кількості багатокутників, усередині яких знаходиться *i*-й багатокутник.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

polydopB.cial:

```
3
3 -2 1 8 9 12 1
3 7 5 6 3 7 4
4 4 3 7 7 9 3 1 2
```

poiydopiz.rez

```
0 2 1
```

11. Головоломка

На столі вряд лежать *N* стопок різнобарвних карток. За один хід з довільної кількості розміщених поруч стопок можна зняти верхні картки певного (лише одного)

кольору. Написати програму САІШЗ, що обчислює мінімальну кількість ходів, потрібних для того, щоб зняти зі столу всі картки.

Перший рядок вхідного файла carciз.ciai містить кількість стопок *IV* (ЛГ > 2). Кожний г-й рядок з наступних ЛГ рядків містить кількість карток *К (К>* 1) в г-й стопці і послідовність з *К* натуральних чисел, що визначають кольори карток в г-й стопці, починаючи з найнижчої картки (1 < ЛГ-ІС < 10 000). Єдиний рядок вихідного файла **carcBiez** повинен містити мінімальну кількість ходів.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

carcia.ciai:
```
2
2 1 2
3 3 1 2
```

carcia.re5
```
3
```

12. Фарбування поля

Дано прямокутне картате поле, що складається з МхЛ[г] клітин. Кожна клітина поля зафарбована в один із шести можливих кольорів, причому верхня ліва і нижня права клітини мають різні кольори. Таким чином поле розбивається на деяку кількість одноколірних областей: дві клітини одного кольору, що мають спільну межу, належать одній області. За першим гравцем закріплена область, що містить ліву верхню клітину поля, за другим - область, що містить його праву нижню клітину. Кожен гравець, виконуючи хід, перефарбовує свою область у будь-який із шести кольорів, в результаті чого до його області приєднуються всі дотичні до неї області обраного кольору, якщо такі існують. Коли після чергового ходу виявляється, що області гравців стикаються одна з одною, гра завершується. Написати програму СОІХЖ, що визначала б мінімальну кількість ходів до завершення гри.

Перший рядок вхідного файла соїог.ciai містить кількість рядків та стовпців поля: *M і N (1 < M, N <* 50). У наступних *M* рядках файла, кожен з яких містить *N* чисел (від 1 до 6), описується, яким чином зафарбовані клітини поля. Перше число у другому рядку файла відповідає кольору лівої верхньої клітини ігрового поля. Кількість одноколірних областей не перевершує 50.

У вихідному файлі соїог.гез потрібно вивести шукану кількість ходів для кожного гравців.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

coïor.ciai
```
4 З
1 2 2
2 2 1
1 4 З
1 3 2
```

coïor.гез
```
3
4
```

13. Торт

Торт, що має форму прямокутної призми з опуклим ЛЛкутником в основі, потрібно розділити на *К* частин рівного об'єму. Дозволяється проводити прямі вертикальні розрізи через увесь торт. Різні розрізи можуть мати спільні точки лише у своїх кінцевих вершинах. Написати програму САКЕ для побудови необхідних *К* - 1 розрізів.

У першому рядку вхідного файла **cake.ciai** містяться згадані вище натуральні числа *К і N (1 < К, N <* 50). Далі записано ЛГ пар дійсних чисел - координати послідовно розташованих вершин ЛГ-кутника.

Кожний рядок вихідного файла **cake.iez** повинен містити чотири числа - координати точок перетину багатокутника відповідного розрізу з основою призми.

Вміст вхідного і вихідного файлів може бути, наприклад, таким:

cake.ciai

```
4 3
2 1
0 0.5
1 0.5
```

cake.iez

```
2 1 0.75 0.5
2 1 0.5 0.5
2 1 0.25 0.5
```

14. Смуга

Задана смуга довжиною 2*к* клітин і шириною в одну клітину. Смугу декілька разів згинають навпіл так, щоб її права половина виявлялася під лівою. Цей процес триває доти, доки згори залишається більше однієї клітини. Написати програму 5ТКІРЕ, яка пронумерує клітини таким чином, що після завершення згинання смуги номери клітин у отриманій колонці будуть впорядковані: 1, 2, 3, 4, ..., 2к.

15. Розріз п-кутника

Дано опуклий многокутник, заданий координатами вершин у порядку їх обходу. Для того щоб розрізати його на трикутники, потрібно провести ЛГ - 2 діагоналей. Написати програму ТКІАМСЬЕЗ, що знаходить мінімальну вартість розрізу. (Вартістю розрізу називається сума довжин усіх використаних діагоналей.) Кількість дій має бути обмежена деяким многочленом, що залежить від *N*.

Перший рядок вхідного файла **triangles.ciai** містить число A^{r}, тобто кількість вершин многокутника. Наступні ЛГ рядків містять по 2 числа - координати ***x*** та ***y*** кожної вершини.

Вихідний файл **triangles.ie5** повинен містити одне значення — мінімальну вартість розрізу.

Додаток

Стандартні модулі СП і Роз в Вогіапсі **Paecal 7,0**

Модуль СМ

До складу модуля включені процедури і функції, що керують виведенням інформації на екран під час роботи відеоадаптера у текстовому режимі, дозволяють зчитувати коди клавіш, які натискаються, а також генерують звукові сигнали.

Вибір текстового режиму

У разі необхідності встановити текстовий режим використовується процедура TexIMocie:

```
Ргосесиге TexiMode(MocIe:могс1);
```

Тут Mocie — код текстового режиму, що визначається константами модуля Сгї, наведеними у табл. 1.

Таблиця 1. Константи модуля Сгі

Ім'я константи	Значення константи	Характеристика режиму
вьі4о	0	Чорно-білий режим, 40 стовпців х 25 рядків
Со40	1	Кольоровий режим, 40 стовпців х 25 рядків
ВИ80	2	Чорно-білий режим, 80 стовпців х 25 рядків
Со80	3	Кольоровий режим, 80 стовпців х 25 рядків
Мопо	7	Використовується з МБА-адаптером
Рогп8х8	256	Використовується для шрифту, що завантажується, в режимах 80х43 і 80х50 з адаптерами **ЕСА** і **УСА**

Керування вікнами

Під час роботи з відеоадаптером у текстовому режимі вікном вважається прямокутна область екрана. Розмір вікна визначається користувачем і не може перевищувати розміру екрана. Процедури виведення даних, керування рядками, курсором і кольором виконують дії відносно поточного вікна. Наприклад, якщо поточним є вікно у 3 символи завширшки, то рядок ' Разсаі' після його виведення на екран виглядатиме так:

```
Раз
саі
```

Процедура Міпсіом(х1,у1,х2,у2:Ьуі:е) створює поточне вікно, лівий верхній кут якого має координати (х1,уі), а правий нижній — (х2,у2). Якщо процедуру Иіпсіом не викликано жодного разу, то поточним вікном вважається весь екран. Очищення поточного вікна здійснює процедура Сі гзсг, що не має параметрів.

Керування рядками

Для керування екранними рядками у текстовому режимі в модулі Сгі означені процедури, наведені у табл. 2.

Таблиця 2. Процедури керування рядками

Ім'я процедури	Призначення	Синтаксис виклику
Іпзііпе	Вставка порожнього рядка перед рядком, на якому знаходиться курсор	Іпзііпе;
ОеПіпе	Видалення рядка, на якому встановлено курсор, із зсувом на одну позицію догори всіх рядків, що знаходились нижче	Оєі1Іпе;
Сігеоі	Видалення вмісту рядка, починаючи від позиції курсору до кінця рядка	СІГЄ0І;

Керування кольором

Процедури, призначені для керування кольором, перелічені у табл. 3.

Таблиця 3. Процедури керування кольором

Ім'я процедури	Призначення	Оголошення
ТехІСої ог	Встановлення кольорів символів. Кольори задаються кодами від 0 до 15 або константами: Ьіаск,..., мііііе. Додаванням константи Ьіпк до коду кольору забезпечується мерехтіння символів.	Ргосесііиге ТехїСоїог (Соїог: Вуіе);
ТехіВаскСгоипсі	Встановлення кольору фону. Код кольору не повинен перевищувати 7. Використання кольорів фону з кодами від 8 до 15 приведе до мерехтіння символів.	Ргосесііиге ТехіВаскбгоипсі (Соїог: Вуїе):

Далі у табл. 4 наведено константи кольорів, які можна встановлювати описаними вище процедурами.

Таблиця 4. Основні кольори та відповідні їм константи

Константа	код	Значення коду	Константа	Код	Значення коду
Віаск	0	Чорний	ИдЬШие	9	синій
Віие	1	Темно-синій	ИдНІдгееп	10	Зелений
Сгееп	2	Темно-зелений	ИдМхуап	11	Блакитний
Суап	3	Темно-блакитний	**ИдНігесІ**	12	Червоний
Кесі	4	Бордовий	ИдНітадепІа	13	Фіолетовий
Маденіа	5	Темно-фіолетовий	Уеііом	14	Жовтий
Втомі	**6**	Рудий	**Шїе**	15	Білий
ИдНІдгау	**7**	Сірий	Віінк	128	Мерехтіння символів
Оагкдгау	8	Темно-сірий			

Приклади використання процедур керування кольором:

```
іехїсоїог (4); {встановлення бордового кольору символів}
Іехїсоїог (4+128): {мерехтіння бордових символів}
ІехїЬаскдгоипсКдгееп): {зелений колір фону }
```

Яскравість символів визначається процедурами I_o\Jli cieo (мінімальна яскравість), Иоті-Уісіео (нормальна яскравість), НідНУісіео (максимальна яскравість). Зауважимо, що ці процедури не мають параметрів.

Керування курсором

У текстовому режимі можна встановлювати курсор у певну позицію в активному вікні, а також визначати координати курсору. З цією метою використовуються процедура та функції, описані у табл. 5.

Таблиця 5. Процедура та функції, **призначені** для керування курсором

Ім'я процедури або функції	Призначення	Оголошення
CoïoXY, процедура	Встановлення курсору в задану позицію поточного вікна, x — номер стовпця, y — номер рядка	processiiire 5oloXY(X. Y: Iniedeг);
MereX, MHereY, функції	Визначення положення курсору	Iinciiop IMereX: Iniedeг: тинсіоп MliereY Iniedeг;

Керування клавіатурою

Функція Keacikey повертає значення введеного з клавіатури символу (значення типу siïar), причому під час її використання зчитаний символ не відображається на екрані. Функція Keyrgezzesi повертає значення Igie, якщо від моменту останнього зчитування символів зі стандартного файла ipriі (що здійснюється процедурами Keaci, KeacJIn та функцією Keacikey) була натиснута будь-яка клавіша, і Iaize - в іншому разі.

Клавіші стандартної клавіатури належать до однієї з трьох категорій: символьні (букви, цифри, інші символи), функціональні (Epiег, Esc, P1...P12, пересунення курсору тощо) і клавіші-модифікатори (Cir!, AII, 5HIJI тощо), В результаті натискання символьної клавіші контролер клавіатури генерує A5CII-код відповідного символу. Під час натискання функціональних клавіш генерується розширений код, що складається з двобайтової послідовності: перший (молодший) байт дорівнює нулю, другий (старший) байт — коду функціональної клавіші. Тому для зчитування коду функціональної клавіші функцію Keacikey слід викликати двічі. Наприклад, після натискання клавіші Esc перший виклик функції Keacikey поверне нуль, а другий виклик — символ з кодом 27.

Керування звуком

Комп'ютерний динамік генерує звукові сигнали, що мають частоту від 27 Гц до 32 000 Гц. Частотою звука та його тривалістю керують процедури ZoinscKHr: Morci), jielayCM3: Morcl) і Mo3oinci. Параметром процедури Zoinci є частота звука (в герцах), а процедури Oeiay — тривалість звука (в мілісекундах). Значення частот, що відповідають нотам першої октави, такі: 523,25 (до), 587,33 (ре), 659,26 (мі), 698,46 (фа), 784,99 (соль), 880,00 (ля), 987,77 (сі). Для переходу на наступну, вищу, октаву потрібно відповідну частоту нижчої октави помножити на 2. Так, нота «до» другої октави має частоту 523,25 x 2 = 1046,50. Процедура Mo3oinci припиняє звучання.

Керування буфером екрана

Буфером екрана називається область пам'яті, до якої записуються дані перед їх виведенням на екран. Буфер складається з декількох ділянок пам'яті, що називаються сторінками. Кожна сторінка містить інформацію про вміст всього екрана, а використання декількох сторінок дає можливість підготувати екранне зображення «у затінку», тобто під час виведення попереднього зображення, а потім вивести його з високою швидкістю. Тому екранний буфер використовується, насамперед, у програмах, що обробляють рухомі зображення.

Буфер екрана для текстового режиму адаптера УСА поділяється на сторінки обсягом 4000 байт кожна — 2000 байт використовується для збереження символів і 2000 байт — для збереження кодів кольорів тексту та фону. Одна сторінка відповідає одному екрану розміром 80x25. Абсолютною адресою нульової сторінки є $B800:$0000, першої — §B900:80000, другої - $BA00:$0000, третьої — $BB00:$0000. Наведемо приклад програми, яка зберігає екранне зображення в буфері, а потім поновлює його.

```
Iype Typeбcгeeп=array[1.4000] oi Ьyie;
var Zcreen: TypeZcreen aЬїoIиIe $B800:$0000;
```

```
    ЬиІег:ТуреЅсгееп;
Ьеgіп
    ЬиІег:=Ѕсгееп; {збереження буфера екрана}

    Зсгееп:=ЬиІег;{відновлення буфера екрана}
епсі.
```

Модуль 005

Підпрограми модуля Б05 виконують певні функції операційної системи. Зокрема, йдеться про функції встановлення і отримання дати й часу, пошуку файлів в каталогах, завантаження та виконання ехе-файлів, обробки переривань тощо.

Функції, що визначають обсяг дискового простору

Нижче перелічені функції, за допомогою яких можна визначити обсяг дискового простору (табл. 6).

Таблиця 6. Функції, що визначають обсяг дискового простору

Ім'я функції	Призначення	Оголошення
•і зкРгее	Визначення обсягу вільної області диску	**Рипсііоп ОізкРгееШгІУЄ: Вуїе):** Іопдіпі;
ОізкЗіге	Визначення обсягу диску	**Рипсііоп ОізкЗігеШгІуе: Вуїе): Іопдіпі;**

Параметрами функцій є номери дисків: **1** — диск А, **2** — диск **В, 3** — диск С тощо. Якщо вказується номер неіснуючого диску, функція повертає значення -і.

Процедури та функції для роботи з файлами

Для роботи з файлами в мові Разсаі використовуються процедури й функції, які описані в табл. 7.

Таблиця 7. Процедури та функції, призначені для роботи з файлами

Ім'я процедури або функції	Призначення	Оголошення
РіпсіРігзІ, процедура	Пошук першого файла із заданим іменем і атрибутом в заданому каталозі	Ргосесіиге РіпсіРігзКРаІІі: РСНаг; АИг: морсі: уаг Р: ТЗеагсіііКес);
РіпсМехІ, процедура	Пошук файла, ім'я, атрибут і каталог якого були зазначені під час попереднього виклику процедури **РІПСІРІГЗІ**	Ргосесіиге РіпсМехШаг Р: ТБеагсИКес);
СеІРАІІг, процедура	Отримання атрибута файла	Ргосесіиге СеІРАИг(уаг Р; уаг АІІг: Уогсі);
ЗеІРАІІг, процедура	Встановлення атрибута файла	Ргосесіиге 5еІРАІІг(уаг Р; АІІг: Иогсі);
Ргеагсії, функція	Пошук файла з заданим іменем в списку каталогів ОІ **П**І зі (елементи списку розділяються символом «;»)	Рипсііоп РзеагсЬСРаІН: РаІНЗІг; ОігИзІ: зігіпд): РаІЬЗІг;
Рзрі і Ъ, процедура	Визначення каталогу, імені файла і розширення імені файла	Ргосесіиге РзрІіКРаІН: РаІІтЗІг; уаг Оіг: ОігЗІг; уаг Мате: №.те5Іг; уаг Ехі: ЕхІБІг);

Семантика параметрів процедур і функцій є такою: Раїїі — ім'я файла, що може бути маскою імені і включати шлях до файла; Аі:і.г — атрибут файла; Р — запис з інформацією про файл; ОігИзІ — список імен каталогів; Ріг — ім'я каталогу; Иапае — ім'я файла; Ехі — розширення імені файла. Типи параметрів РаІЬ, АИг і Оігііізі є стандартними, а типи решти параметрів означені в модулі Р05.

Атрибути файла задаються такими константами: РеасіогГіу (тільки для читання), сіі гесіогу (каталог), НісЮеп (прихований), агсИіуе (архівний), з у зііі е (системний), а п уііі е (довільний).

Наведемо оголошення записів, що використовуються в процедурах і функціях, призначених для обробки файлів.

```
Type
  TSearcIiKec = recorci
    Pi ïï: arrau[1. 21] oi Вуїе;
    АИг; Вуїе;  Ti te: Іопдіпі;
    Зі ге: І о гіді пі;
    Mare- arrau[0 .12] oT Cipar;
  epci;
  TPTleKec = recorci
    HanciIe: Horci'; Mocie; Morй;
    КєсЗіге: MorcI;
    Рпуаіе; arrau[ 1. .26] oi Вуїе;
    IIзегOaIa; arrau[I .16] от Вуїе;
    Mare: arrau[0 .79] от CI'iar;
  epci;
  TTєхРЧЄС = recorci
    Напоїе: MorcI: Mocie; иогсі;
    ВиІЗіге; MorcI: Prіyaie: MorcI;
    ВиІРоз; Mcri; ВиГЕпсІ: MorcI;
    ВиІРІг: РІехІВиІ
    ОрепРипс. Роіпіег; Сіозегипс: Роїпіег;
    ІпОиІРипс: Роїпіег
    РИзЬРипс: Роіпіег;
    'і з є г 0 а І а: а г г а у Г1 ..16] о І Вуїе;
    Nanie: arrau[0. 79] оі CHar;
    ВиїїІег; TTехІВиІ;
  epci;
```

Процедури обробки системної дати і часу

Мова Разса! дає можливість визначати та встановлювати системну дату та час. Для цього призначені процедури, перелічені у табл. 8,

Таблиця 8. **Процедури обробки** системної дати і часу

Ім'я процедури	Призначення	**Оголошення**
СеЮаІе	Отримання системної дати	ргосесіиге СеМаІе(уаг Уеаг, МопІН, Оау, ОауОПмеек; Иогсі);
ЗєЮаІє	Встановлення системної дати	ргосесіиге ЗеЮаІе(Уеаг, МопШі, йау: Могсі);
беІТіше	Отримання системного часу	ргосесіиге ЙеІТі те (уаг Ноиг, Міпиіе. Зесопсі. ЗесІОО: MorcI);
ЗеІТіте	Встановлення системного часу	ргосесіиге ЗеІТітеСНоиг, Міпиіе, Зесопсі ЗесІОО: Моrо);

Семантика параметрів процедур є такою: Уеаг – рік, МопШИ — місяць, Оау — день місяця, ОауОтиєек — номер дня тижня, Ноиг - година, Міпиіе — хвилина, Зесопсі — секунда, ЗесІОО — сота частка секунди.

Процедури для роботи з перериваннями

У модулі БОЗ оголошено декілька процедур, призначених для безпосереднього керування перериваннями (табл. 9).

Таблиця 9. Процедури для роботи з перериваннями

Ім'я процедури	Призначення	Оголошення
IPIГ	Виконання переривання з заданим номером	Ргосесіиге I л MI п Ш: Вуїе; уаг Кедз: Кеді зіе г з);
СеііпіУес	Визначення за номером переривання його вектора	Ргосесіиге иеІІпІ¥ес(IпIМо: Вуїе; уаг Iесіог; Poiпіег);
ЗеІІпІУес	Встановлення нового вектора для переривання із заданим номером	Ргосесіиге 2еІІпІ¥ е с (IпШ: Вуїе; Уесіог: Poiпіег):

Процедури мають такі параметри: ІпіИо — номер переривання, Кедз - регістри процесора, Уесіог — вектор переривання, тобто адреса процедури обробки переривання.

Процедури та функції обробки процесів

Програма, що завантажена до оперативної пам'яті з метою виконання, є процесом. Процедури обробки процесів (табл. 10) можуть завантажувати програми, що зберігаються в файлах, імена яких мають розширення сот або ехе, і передавати параметри до програм, що завантажуються, аналогічно тому, як це робиться за допомогою командного рядка МЗ-БОЗ.

Таблиця 10. Процедури та функції обробки процесів

Ім'я процедури або функції	Призначення	Оголошення
Кеер, процедура	Завершення резидентної програми	Ргсseаиге КеерСЕхНСосіє: Нопі);
Exec, процедура	Виконання процесу	Ргосесіиге Exec (Раїіі, СтсІІі пе: зігіпд);
ЗмарУесІогз, процедура	Збереження і поновлення значень векторів переривань	Ргосесіиге ЗмарУесІогз;
ОозЕхіІСосІе, процедура	Визначення коду завершення підпроцесу	Рипсііоп ОозЕхіІСосіе: догсі;

Процедури та функції мають такі параметри: ЕхіІСойе — код завершення програми, Раїіі — повне ім'я файла, СшсНіпе — команда БОЗ, яка має виконувати певну програму.

Оскільки більшість програм змінює стандартні значення векторів переривань, перед запуском підпроцесу ці стандартні значення мають бути поновлені, а після його виконання будуть повернені ті значення, що були встановлені головною програмою. Ці дії виконує процедура ЗмарУесІогз, яку слід викликати безпосередньо до і після виклику процедури Exec.

У програмі, що викликає підпроцеси, потрібно вказати директиву компілятора {$M обсяг_стеку, **мінімальний**_обсяг_купи, максимальний_обсяг_купи,, яка встановлює обсяги стекової і динамічної пам'яті для процесів. За замовчуванням обсяг динамічної пам'яті процесу дорівнює обсягу всієї доступної програмам пам'яті, **і** тому завантаження підпроцесу можливе лише за умови, що цей обсяг буде зменшений.

В результаті виконання підпроцесу змінній ОозЕгтог присвоюється код, за яким можна визначити причину успішного або аварійного завершення підпроцесу. Код 0 означає успішне завершення, інші значення змінної ОозЕгтог — аварійне.

Література

1. *Абрамов С. А., Гнездилова Г. Г., Капустина Е.* Я., Селюн М. И. Задачи по программированию. - М.: Наука, 1988. - 280 с.
2. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вичислительних алгоритмов.— М.: Мир, 1979. - 536 с.
3. *Бородич Ю. С., Вальвачев А.* Я, *Кузьмич А. И.* Паскаль для персональних компьютеров. — Минск: Выш. шк, 1991. — 364 с.
4. *Буч Г.* Обьектно-ориентированньш анализ и проектирование с примерами приложений на С++. — 2-е изд. — М.: Бином, СПб.: Невский диалект, 1999. — 560 с.
5. *Вирт* Я. Алгоритми + структури даннн - программн. — М.: Мир, 1985. — 406 с.
6. *Воєводин В. В.* Линейная алгебра М.: Наука, 1980. — 400 с.
7. *Зеленяк О. ТІ,* Практикум программирования на ТигЬо Разсаі: Задачи, алгоритми и решения. - СПб.: ДиаСофтЮП, 2003. - 320 с.
8. *Зелковиц М., Шоу А., Гзннон Дж.* Принципи разработки программного обеспечения/Пер. с англ. - М.: Мир, 1982. - 368 с.
9. *Иванов Б.* Я. Дискретная математика. Алгоритми и программн. — М.: Лаб. базових знаний, 2002. - 228 с.
10. Інформатика: Комп'ютерна техніка. Комп'ютерні технології/За ред. О. І. Пушкаря. — К: Видав, центр «Академія», 2002. — 704 с.
11. *Йодан З.* Структурное программирование и конструирование програми. — М.: Мир, 1979. — 416 с.
12. *КнутД.* Искусство программирования. — Т. 3: Сортировка и поиск. — М.: Вильямс, 2004. — 703 с.
13. *Кормен Т., Лейзерсон Ч,, Ривест Р.* Алгоритми: построение и анализ. — М.: МЦНМО, 2001. — 960 с.
14. *Лингер Р., Миллс Х., Уитт Б.* Теория и практика структурного программирования. — М.: Мир, 1982. - 406 с.
15. *Липский В.* Комбинаторика для программистов. — М.: Мир, 1988. — 213 с.
16. *Майника З.* Алгоритми оптимизации на сетях и графах. — М.: Мир, 1981. — 323 с.
17. *Мандельброт Б.* Фрактальная геометрия природи/Пер. с англ. - М.: Ин-т компьютер. исслед., 2002. - 656 с.
18. *Марченко А. И., Марченко Л. А.* Программирование в среде ТигЬо Разсаі 7.0, — К: ВЕК+, 2000,- 464 с.
19. *Немнюгин С. А.* ТигЬо Разсаі: Учебник. — СПб.: Питер, 2003. — 468 с.
20. *Немнюгин С. А.* ТигЬо Разсаі: Практикум. — СПб.: Питер, 2003. — 256 с.
21. *Новиков Ф. А.* Дискретная математика для программистов. — СПб.: Питер, 2003. — 304 с.
22. *Окулов С. М.* Программирование в алгоритмах. — М.: Бином, Лаб. базових знаний, 2002. — 341 с.
23. *Пильщиков В. Н.* Язик Паскаль: Упражнения и задачи. — М.: Науч. мир, 2003. — 224 с.
24. *Рейнгольд З., Нивергельт Ю., Део* Я. Комбинаторние алгоритми: Теория и практика. — М.: Мир, 1980. - 476 с.
25. *Сердюченко В. Я.* Розробка алгоритмів та програмування мовою ТигЬо Разсаі. — Харків: Паритет, 1995. - 352 с.
26. *Ставровский А. Б.* Турбо Паскаль 7.0. — К.: Издат. группа ВНУ, 2000. — 400 с.
27. *Сухарев М.* ТигЬо Разсаі 7.0. Теория и практика программирования. — СПб.: Наука и техника, 2003. - 576 с.
28. Турбо Паскаль 7.0: Самоучитель. — СПб.: Питер; К: Издат. группа ВНУ, 2002. — 416 с.
29. *Фаронов В. В.* Турбо Паскаль 7.0. Начальний курс. — М.: Нолидж, 1999. — 616 с.
30. *Шелест В. Д.* Программирование. — СПб.: БХВ Петербург, 2001. — 592 с.
31. *Узлстид С.* Фрактали и вейвлети для сжатия изображений в действии. — М.: Триумф, 2003. - 320 с.

Алфавітний покажчик

Н

Р

С

Навчальне видання

Ковалюк Тетяна Володимирівна

ОСНОВИ ПРОГРАМУВАННЯ

Підручник

Керівник проекту І. О. Завадський

Редактор Л. Д. Кожем'яко

Коректор І. В. Карпишенко

Комп'ютерна верстка З. В. Лобач

ТОВ «Видавнича група ВНУ»
Свідоцтво про внесення до Державного реєстру
суб'єктів видавничої справи України
серія ДК №175 від 13.09.2000 р.

Підписано до друку 09.08.05. Формат 70x100 1/ц.
Папір офсетний. Гарнітура РеІегзЬиг§. Друк офсетний.
Ум. друк. арк. 30,96. Обл.-вид. арк. 29,03.
Наклад 3000 прим. Зам. №30-143.

Виготовлено в ТОВ «Освітня книга»,
м. Київ, вул. Орловська, 2/7, оф. 6.
Свідоцтво про внесення до Державного реєстру
суб'єктів видавничої справи України
серія ДК №2245 від 26.07.2005 р.

Т. В. Ковалюк

ОСНОВИ ПРОГРАМУВАННЯ

Метою створення підручника є сис~е»."-е "р,нтов-е й доступне викладення найважгив _их "о-я-= концепцій і метод:в структурного програмування. Підручник має практичне спрямуван-я. V стить численні приклади розв яза--я зада- з також зада- вправи і завдання для самос- йно ообст/ с^де-- в Спосіб подання матер агу є методи-чо вивіреним неодноразово а-ообсва-,".' а- .'.'.лює найкоащ здобутки класико' -оади_ зиоадан-я даного предмета

Базовий курс для студентів виних навчальних закладів, як навчаються за напрямами: «Комп'ютерні науки», «Комп'ютерна інженерія», «Прикладна математика», ютеризовані системи, автоматика і управління».

Детально розглянуто такі теми:

- **базові поня~я алгори-м за. мови Разса**
- **методологія сто.кту:-ос с П[**
- **використання ~д~ос":** <____.
- **статичні та ди-а1.' -- : ктури дз-их їх обробки:**
- **методи програмного розв'язання захливих математич-их 1 комб ~а-орних ззд з-**

Зміст підручників серії відповідає навчальним планам з інформатики, що використовуються у провідних вищих навчальних закладах України

Ковалюк Тетяна Володимирівна — онд/дат -з* - наук, доцент, вчений се-се-а^ -а.•:зс-'.'е-;ди-- з комп'ютерних -а,- '.' -істерства осз ~н -а.-и .• розробник галузевих стандарт з ви_до с:з ~и .•••: автор 46 наукових сесед яких дза ~ д: . -- з грифом Міністерства осе ти -а,.-и -и Зд наукових інтеоес з — "ех-с.~о" навчальних прогоа1.'

I5BM 966552138-"