**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**«ІНСТИТУТ ПРОМИСЛОВИХ ТА БІЗНЕС ТЕХНОЛОГІЙ»**
**УКРАЇНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ НАУКИ І ТЕХНОЛОГІЙ**

**Л.М. БАНДОРІНА, Т.О. КЛИМКОВИЧ, К.О. УДАЧИНА**

# ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ

**Дніпро 2022**

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**«ІНСТИТУТ ПРОМИСЛОВИХ ТА БІЗНЕС ТЕХНОЛОГІЙ»**
**УКРАЇНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ НАУКИ І ТЕХНОЛОГІЙ**

**Л.М. БАНДОРІНА, Т.О. КЛИМКОВИЧ, К.О. УДАЧИНА**

# ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ

Друкується за Планом видань навчальної та методичної літератури,
затвердженим Вченою радою ННІ ІПБТ УДУНТ
Протокол № 1 від 24.01.2022

**Дніпро 2022**

УДК 004.76

Бандоріна Л.М., Климкович Т.О., Удачина К.О. Основи алгоритмізації та програмування : навч. посібник. УДУНТ, 2022. 158 с.

Наведено навчальні матеріали з основ алгоритмізації і програмування, застосування конструкцій мови С++, розробки масивів, функцій, управляючих структур тощо, представлено концепції і термінологію структурного і об'єктно-орієнтованого програмування.

Призначений для студентів спеціальностей 051 – Економіка, 126 – Інформаційні системи та технології (бакалаврський рівень).

Друкується за авторською редакцією.

Укладачі: Л.М. Бандоріна, канд. екон. наук, доц.
Т.О. Климкович, ст. викл.
К.О. Удачина, канд. екон. наук, доц.

Відповідальна за випуск Л.М. Бандоріна, канд. екон. наук, доц.

Рецензенти: В.А. Турчина, канд. фіз.-мат. наук, доцент, завідувач кафедри обчислювальної математики та математичної кібернетики Дніпровського національного університету імені Олеся Гончара

Е.В. Терещенко канд. фіз.-мат. наук., доцент, доцент кафедри системного аналізу та обчислювальної математики НУ «Запорізька політехніка»

Підписано до друку 18.04.2022. Формат 60х84 1/16. Папір друк. Друк плоский. Облік.-вид. арк. 9,29. Умов. друк. арк. 9,17. Замовлення № 42

Навчально-науковий інститут
«Інститут промислових та бізнес технологій»
Українського державного університету науки і технологій 49005, м. Дніпро, пр. Гагаріна, 4
Редакційно-видавничий відділ ННІ ІПБТ УДУНТ

## ЗМІСТ

ПЕРЕДМОВА ........ 5
1 Типові алгоритмічні конструкції і введення у програмування ........ 6
1.1 Поняття алгоритму та його властивості ........ 6
1.2 Способи опису алгоритмів ........ 7
1.3 Графічний опис алгоритмів ........ 7
1.4 Різновиди структур алгоритмів ........ 9
1.5 Приклади створення блок-схем алгоритмів ........ 12
1.6 Розміщення даних та програм у пам'яті ПЕОМ ........ 16
1.7 Теорія структурного програмування ........ 19
1.8 Програмування на основі мови С++ ........ 22
1.9 Синтаксис операторів мови С++ ........ 42
2 Алгоритми та програмування умовних процесів ........ 45
2.1 Оператори переходу (умовного та безумовного) ........ 45
2.2 Оператор if – else ........ 45
2.3 Оператор вибору альтернатив switch ........ 48
3 Алгоритми та програмування циклічних процесів ........ 53
3.1 Оператор циклу while ........ 53
3.2 Оператор циклу for ........ 54
3.3 Оператор циклу do while ........ 56
4 Користувацькі функції ........ 58
4.1 Призначення функцій ........ 58
4.2 Створення і використання простої функції ........ 60
4.3 Оголошення і визначення функцій ........ 61
4.4 Аргументи функції. Формальні і фактичні аргументи ........ 63
4.5 Виклики функцій ........ 65
4.6 Опис типів функції ........ 65
4.7 Покажчики ........ 66
5 Алгоритми і програми обробки числових масивів інформації ........ 70
5.1 Обробка одновимірних числових масивів інформації ........ 70
5.2 Обробка багатовимірних числових масивів інформації ........ 74
5.3 Використання покажчиків при роботі з масивами ........ 76
5.4 Створення динамічних масивів в С ++ ........ 79
6 Алгоритми впорядкування інформації ........ 86
6.1 Упорядкування даних методом вибору ........ 86
6.2 Упорядкування даних методом попарної перестановки ........ 88
6.3 Упорядкування даних методом Шелла ........ 88
6.4 Обробка символьних масивів. Функції над рядками ........ 91
7 Структури ........ 96
8 Файли ........ 103
9 Об’єктно-орієнтоване програмування ........ 108
9.1 Основні поняття об’єктно-орієнтованого програмування (ООП) ........ 108

9.2 Класи та структури С++ як засоби реалізації ОПП ................................. 110
9.3 Обмеження доступу до атрибутів та методів класу ................................. 113
9.4 Конструктори і деструктори ................................................................. 117
9.5 Спадкування ........................................................................................... 125
9.6 Поліморфізм ........................................................................................... 129
9.7 Абстрактні класи ..................................................................................... 135
9.8 Класи вводу-виводу ................................................................................ 136
9.9 Файлові потоки вводу-виводу в С++ ...................................................... 140
9.10 Тести до розділу «Об'єктно-орієнтовне програмування» ....................... 144
СПИСОК ЛІТЕРАТУРИ ................................................................................ 149
Додаток А .................................................................................................... 151
Відповіді на тести ........................................................................................ 151

# ПЕРЕДМОВА

Метою дисципліни «Основи алгоритмізації та програмування» є формування логічного мислення і знання основних понять алгоритмізації та техніки застосування у програмуванні базових алгоритмічних структур.

Під час вивчення дисципліни студент має:

– вивчити теоретичні основи алгоритмізації та структурного програмування;

– вивчити синтаксис та базові конструкції мови С++;

– набути вмінь з розробки алгоритмів і складання програм мовою С++, забезпечуючи вирішення економічних задач;

– окремий розділ присвячено засобам об’єктно-орієнтованого програмування;

– набути вмінь застосовувати сучасне інструментальне програмне забезпечення для складання та налагодження програм за допомогою вбудованих інструментальних засобів.

Навчальний посібник охоплює майже всі теми дисципліни «Основи алгоритмізації та програмування», що викладається для студентів спеціальностей 051 – Економіка та 126 – Інформаційні системи та технології.

Кожна тема навчального посібника містить:

– основний текст, в якому активно використовуються приклади застосування викладеного матеріалу;

– запитання та тести для самоконтролю знань і практичного освоєння матеріалу.

Усі наведені приклади програм реалізовані у середовищі Microsoft Visual Studio у консольному додатку.

# 1 ТИПОВІ АЛГОРИТМІЧНІ КОНСТРУКЦІЇ І ВВЕДЕННЯ У ПРОГРАМУВАННЯ

## *1.1 Поняття алгоритму та його властивості*

Під алгоритмізацією розуміється зведення задачі до послідовних етапів дій так, що результати попередніх дій використовуються при виконанні наступних. При цьому мають бути чітко зазначені як зміст кожного етапу, і порядок виконання етапів. Окремий етап алгоритму є або інше, більш просте завдання, алгоритм вирішення якого відомий (розроблений заздалегідь), або має бути досить простим і зрозумілим без пояснень. Тобто алгоритм – це послідовність точно визначених дій, що однозначно призводять до вирішення поставленого завдання.

Вирішення будь-якої задачі можна розбити на наступні етапи: розробка алгоритму розв'язання задачі, складання програми розв'язання задачі алгоритмічною мовою, введення програми, налагодження програми (виправлення помилок), виконання програми, аналіз отриманих результатів.

Можна сказати, що алгоритм визначає процес перетворення вихідних даних на результати, оскільки для вирішення будь-якого завдання необхідно:

– ввести вихідні дані;

– перетворити вихідні дані на результати (вихідні дані);

– вивести результати.

Алгоритми мають такі властивості:

– *дискретність* – тобто розчленованість алгоритму на зручні, зрозумілі, доступні частини;

– *масовість* (типовість) – можливість використання для інших задач того ж класу;

– *результативність (кінцевість)* – отримання результату завжди, алгоритм вирішує поставлене завдання за кінцеве число кроків;

– *детермінованість* – визначеність, однозначність результату при заданих похідних даних.

Процес алгоритмізації передує процесу програмування.

## 1.2 Способи опису алгоритмів

Існує кілька способів опису алгоритмів. Найбільш поширеними є словесний і графічний опис алгоритму.

Словесний опис алгоритму розглянемо на конкретному прикладі: нехай необхідно знайти найбільший спільний дільник для двох цілих позитивних чисел a і b.

1) Порівняти a і b. Якщо a <b, то задати d = a; m = b, інакше d = b і m = a.

2) Розділити m на d. Позначити залишок від ділення r.

3) Якщо d = 0, то це шукане число. Закінчити обчислення. Інакше перейти до пункту 4.

4) Замінити значення m значенням d.

5) Замінити d значення значенням r.

6) Перейти до пункту 2.

Тут алгоритм описаний за допомогою природної мови, а об'єкти обробки, є числами, що позначені літерами.

Наприклад, розглянемо алгоритм вирішення квадратного рівняння виду $a*x^2 + b*x + c = 0$:

1) обчислити D = b*b - 4*a*c;

2) якщо D <0, то перейти до 4;

3) обчислити: $X_1 = (-b + \sqrt{D})/(2*a)$; $X_2 = (-b - \sqrt{D})/(2*a)$;

4) кінець.

## 1.3 Графічний опис алгоритмів

Подання алгоритму у вигляді схеми, що складається з послідовності геометричних фігур (блоків, символів), кожен з яких відображає зміст чергового кроку алгоритму, називається графічним описом алгоритму. Усередині фігур коротко записують дію, яка виконується в цьому блоці.

***Блок-схемою*** називається графічне зображення логічної структури алгоритму, в якому кожний етап процесу переробки даних подається у вигляді блоків, що мають визначену конфігурацію залежно від операцій, що виконуються. Графічні символи, їх розміри, а також правила побудови блок-схем визначені державними стандартами.

За призначенням та характером відображуваних операцій символи поділяються на основні й допоміжні. Повний список включає 42 символи. Найбільш часто використовувані графічні символи наведені у таблиці 1.1.

Правила зображення фігур зведені в єдину систему програмної документації [1]. Кожний блок, що має замкнутий контур, вписується в прямокутник зі сторонами *a* та *b* (рисунок.1.1). Розмір *a* слід обирати з ряду 10, 15, 20 мм. Допускається збільшення розміру *a* на число, кратне 5. Розмір *b* дорівнює 1,5 розміру *a*. В деяких випадках допускається співвідношення $b = 2a$ [2].

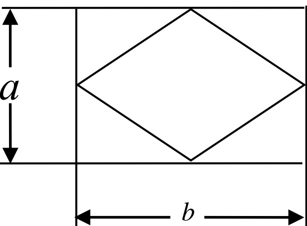



Рисунок 1.1 – Розмір блоків блок-схем

Таблиця 1.1 – Основні символи блок-схем

| Назва символу (блоку) | Графічне відображення | Призначення |
|---|---|---|
| Процес | | Виконання операції або групи операцій, в результаті яких змінюються значення, форма подання або розміщення даних. Усередині символу або ж у вигляді коментарю на природній мові або у вигляді формули записуються дії, які здійснюються при виконанні операції або групи операцій. |
| Розв'язання | | Вибір напрямку виконання алгоритму в залежності від умови. |
| Введення-виведення | | Введення або виведення даних незалежно від фізичного носія. |
| Документ | | Введення-виведення даних, носієм яких служить папір. |
| Пуск - останов | | Початок, кінець, переривання процесу обробки даних або виконання програми. |

| | | |
|---|---|---|
| Зумовлений процес | | Використання раніше створених і окремо описаних алгоритмів або програм (процедур, функцій, програмних модулів). |
| Модифікація | | Виконання дій, що змінюють команди або групу команд, що змінюють програму. Використовується для зображення структури ЦИКЛ З ПАРАМЕТРОМ. |
| З’єднувач | | Вказівка зв’язку між перерваними лініями потоку в межах однієї сторінки. |
| Міжсторін-ковий з’єднувач | | Вказівка зв’язку між роз’єднаними частинами схем алгоритмів та програм, розташованих на різних аркушах. |
| Коментар | | Зв'язок між елементами схеми і поясненнями. Дозволяє включати в блок-схему пояснення, формули та іншу інформацію. |
| Лінії потоку | | Вказівка послідовності зв’язків між символами. |

Правила побудови блок-схем:

1. Блок-схема вибудовується в одному напрямку: або зверху вниз, або зліва направо.

2. Лінії потоку повинні бути паралельними лініям зовнішньої рамки блок-схеми (межам листа, на якому зображена блок-схема).

3. Всі повороти ліній потоку виконуються під кутом 90 градусів.

4. Напрямок лінії потоку зверху вниз і зліва направо приймається за основний і стрілками не позначається, в інших випадках напрямок лінії потоку позначається стрілками.

5. Якщо блок-схема складається з декількох частин, розташованих на одній сторінці, то лінія потоку однієї частини закінчується символом «з'єднувач», а лінія потоку на продовженні блок-схеми починається з цього ж символу. Усередині символів «з'єднувач» ставляться однакові порядкові номери, що відповідає розірваній лінії потоку.

### *1.4 Різновиди структур алгоритмів*

Розрізняють наступні структури алгоритмів:

- послідовність;
- розгалуження;
- цикли.

Блок-схеми структурних алгоритмів – різні комбінації трьох базових структур: ***послідовності***, ***розгалуження*** й ***циклу***. Це необхідний і достатній набір структур для складання блок-схем будь-яких задач.

***Послідовність*** (або ***слідування)*** – керуюча структура, яка означає, що дві або більше дій слід виконати послідовно одна за одною. В кожному блоці послідовності може бути проста дія або сукупність складних обчислень та перетворень. Графічну інтерпретацію послідовності показано на рисунку 1.2.

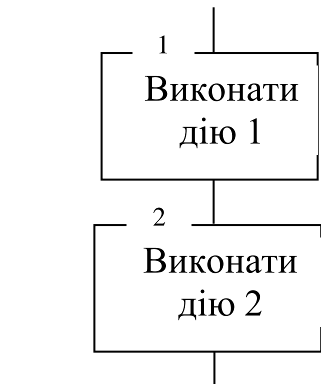



Рисунок 1.2 – Структура послідовності

***Розгалуження*** – керуюча структура, що організує виконання лише однієї з двох зазначених дій в залежності від правдивості деякої умови.

Умова – це питання, що має два варіанти відповіді: так чи ні. Запис розгалуження виконується в двох формах: повної та неповної. Повна й неповна форми структури розгалуження наведено на рисунках 1.3 і 1.4 відповідно.

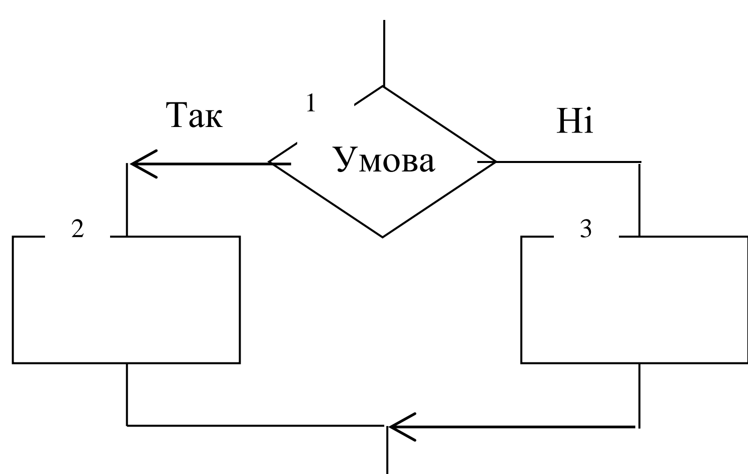



Рисунок 1.3 – Повна форма структури розгалуження

У повній формі структури розгалуження (див. рис. 1.3) залежно від результату перевірки умови виконується один із блоків: 2 або 3.

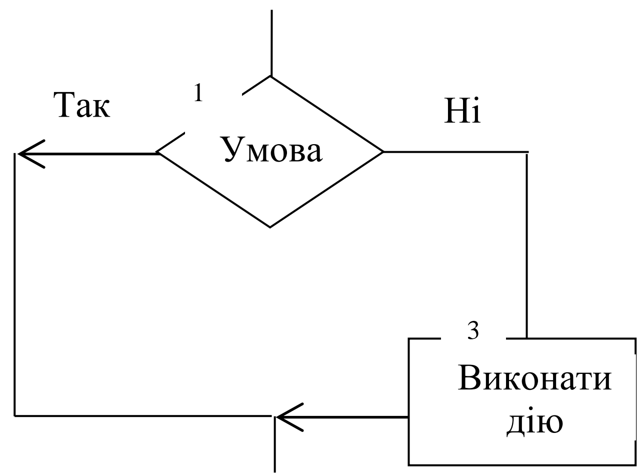



Рисунок 1.4 – Неповна форма структури розгалуження

Циклічна структура – це структура, що організує виконання деяких дій багаторазово. Тобто, організоване повторення деякої послідовності дій називається ***циклом***. При повторенні дій зазвичай використовуються нові значення вхідних даних. Один прохід циклу називається ітерацією.

До структури циклу входять дві основні частини:

1) перевірка умови продовження виконання циклу;

2) власне повторювання частини дій.

Перевірка умови продовження виконання циклу виконується на кожній ітерації або на початку циклу (з передумовою), або в кінці циклу (з постумовою).

Геометричну інтерпретацію різних варіантів циклу: циклу з передумовою і циклу з постумовою наведено на рисунках 1.6 і 1.7 відповідно.

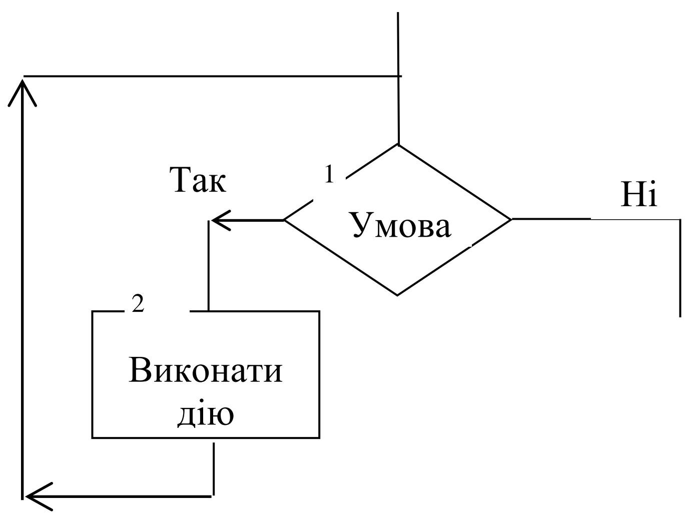



Рисунок 1.6 – Схема циклу з передумовою

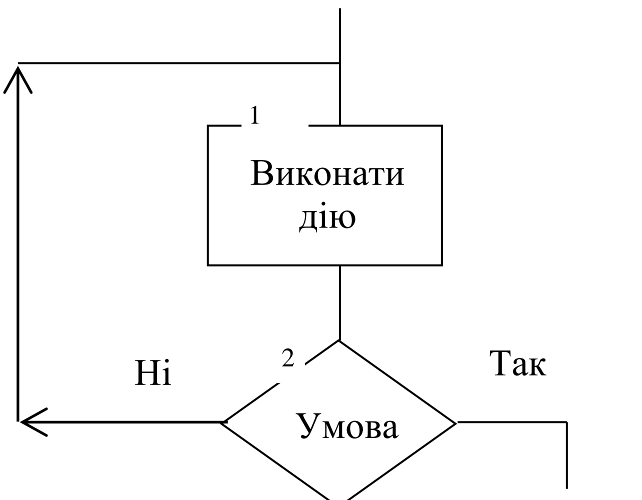



Рисунок 1.7 – Схема циклу з постумовою

Багато з реально існуючих алгоритмів мають змішаний характер, тобто можуть містити лінійні ділянки, розгалуження, цикли. У зв'язку з цим складання алгоритмів відразу в закінченій формі ускладнено. Тому для розробки складних алгоритмів рекомендується використовувати спадне проєктування програм (метод покрокової деталізації, метод послідовних уточнень). Його суть: спочатку продумується загальна структура алгоритму, без детального опрацювання його окремих частин. Далі опрацьовуються окремі блоки, що не деталізовані на попередньому кроці. Таким чином, на кожному кроці розробки уточнюється реалізація фрагмента алгоритму, тобто вирішується простіше завдання.

### *1.5 Приклади створення блок-схем алгоритмів*

*Приклад 1.1*. Розробити алгоритм для обчислення значення функції ***B*** коли **z**=15, ***x*** – неціле число, що вводиться з клавіатури:

$$B = (z\sqrt{x} + x^2) \cdot e^x,$$

Алгоритм обчислення даної функції є лінійним (використовується тільки структура послідовності), оскільки хід обчислювального процесу не залежить від будь-яких умов, і наведений на рисунку 1.8.

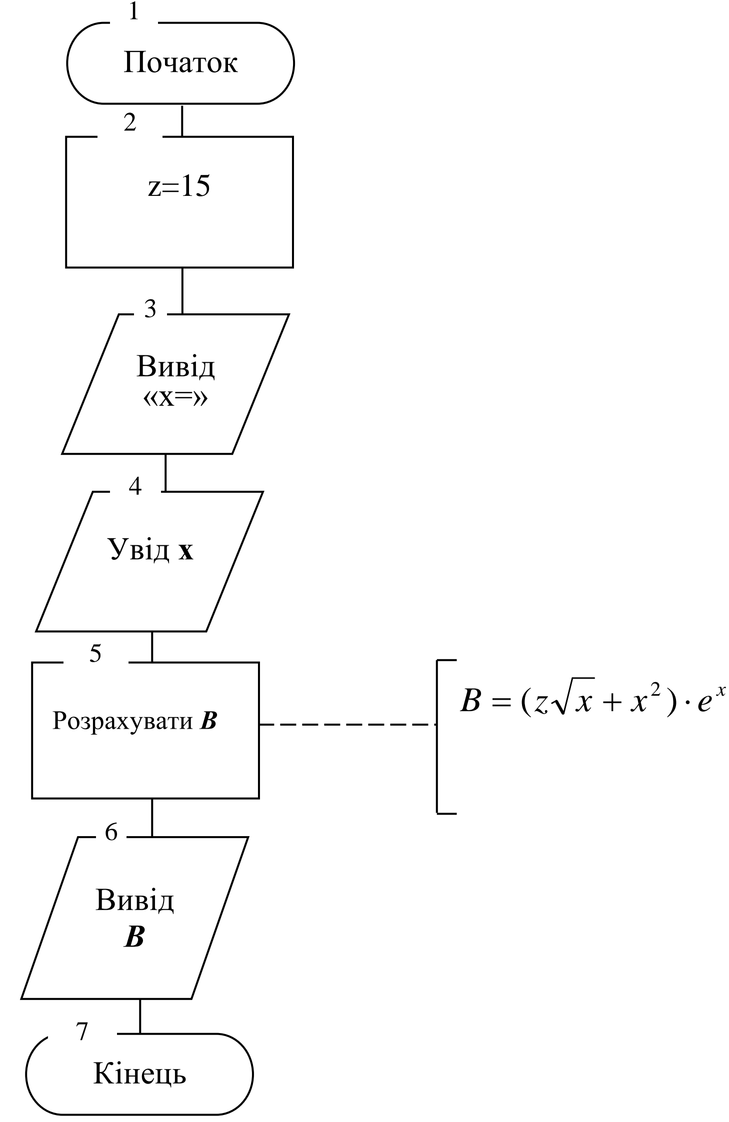



Рисунок 1.8 – Лінійний алгоритм

*Приклад 1.2*. Обчислити ***y****,* де ***x*** – ціле число, яке необхідно ввести з клавіатури.

$$y = \begin{cases} y = 9x^2 + 27x - 1.25, & \text{якщо } -5 \le x < 5; \\ \sqrt{\dfrac{3*|x|}{4}}, & \text{якщо } x < -5; \\ |x - 45|, & \text{якщо } x \ge 5. \end{cases}$$

Алгоритм обчислення ***y*** містить структуру розгалуження, оскільки хід обчислювання ***y*** залежить від умов, і наведений на рисунку 1.9.

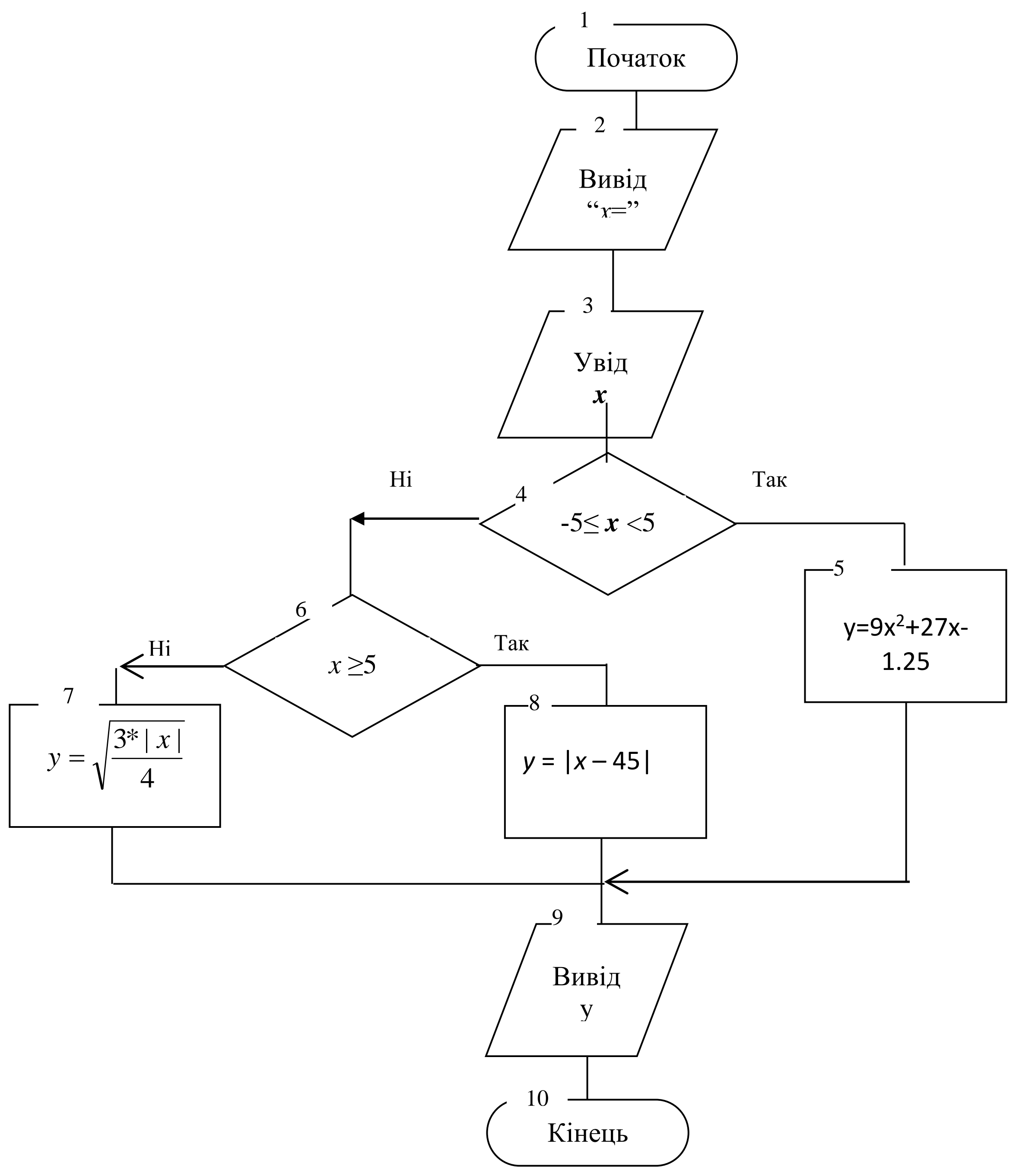



Рисунок 1.9 – Алгоритм з розгалуженням

*Приклад 1.3*. Обчислити F*,* де A=3.5, B і *x* – цілі числа, які необхідно ввести з клавіатури.

$$\boldsymbol{F} = \sum_{k=5}^{10} \frac{\boldsymbol{Ax} + \boldsymbol{B}}{\boldsymbol{k}}.$$

Алгоритм обчислення ***F*** містить цикл з передумовою: блоки 7, 9, 10, 12.

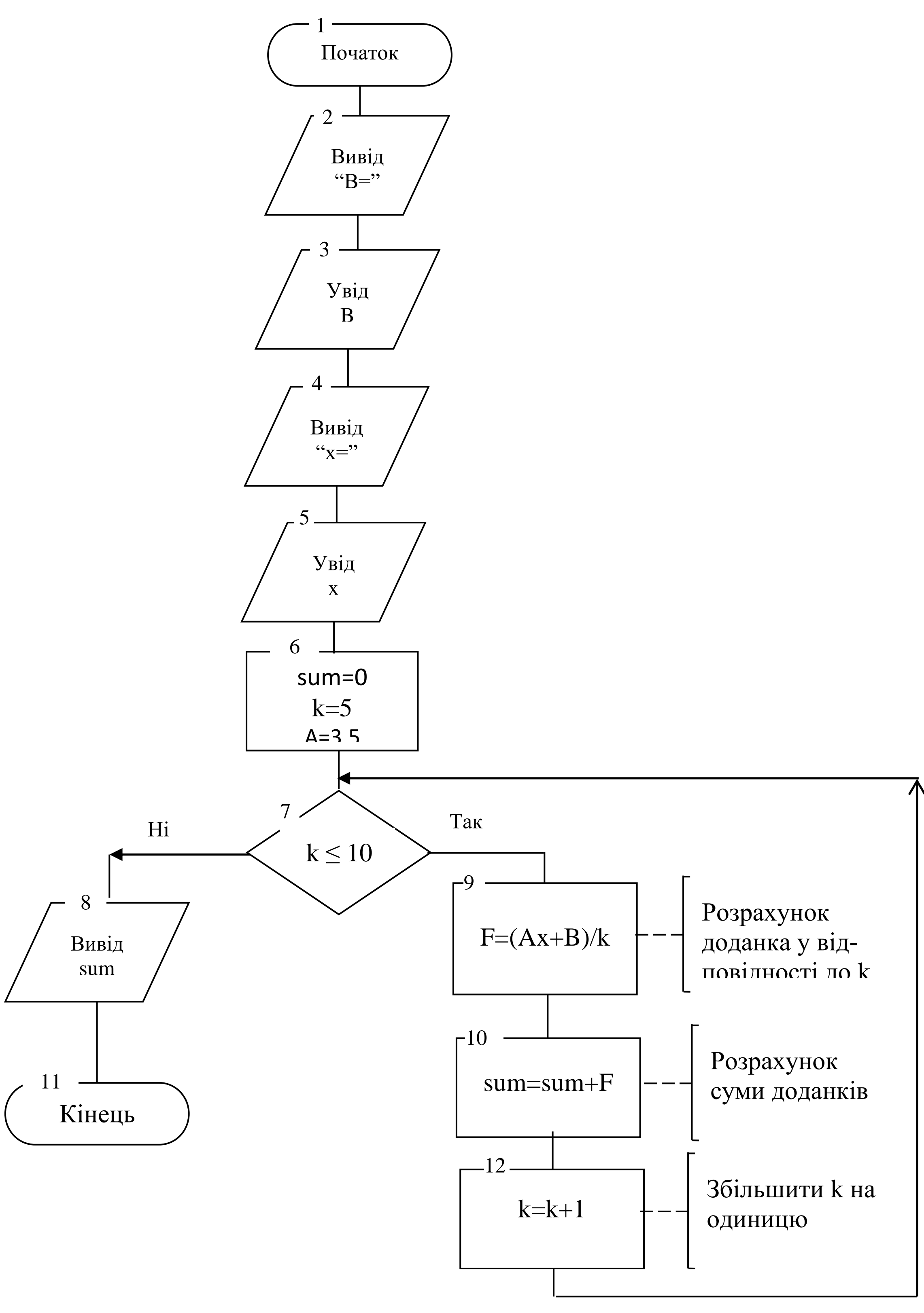



Рисунок 1.10 – Алгоритм з циклом з передумовою

## *1.6 Розміщення даних та програм у пам'яті ПЕОМ*

Мови програмування – система позначень для точного опису алгоритму, який треба виконати за допомогою комп’ютера.

Мова програмування, як і будь-яка інша мова, являє собою набір символів (алфавіт), систему правил складання базових конструкцій мови (синтаксис) та правила тлумачення конструкцій мови (семантика). Ця система позначень і правил призначена для одноманітного і точного запису алгоритму. Алфавіт, синтаксис і семантика – три основні складові мов програмування.

Алгоритм, записаний мовою програмування, називається програмою.

Інакше кажучи, *програма* – це послідовність команд (інструкцій), які містяться у пам'яті ПЕОМ та виконуються процесором у зазначеному порядку.

Дані та програми під час роботи ПЕОМ розміщуються в оперативній пам'яті, яка є послідовністю пронумерованих комірок. За вказаним номером процесор знаходить потрібну комірку, тому номер комірки називається її *адресою*. Мінімальна адресована комірка (відповідно до стандарту IBM) складається з 8 двійкових позицій, тобто у кожну позицію можуть бути записані або 0, або 1.

Обсяг інформації, що міститься в одну двійкову позицію, називається *бітом*.

Обсяг інформації, що дорівнює 8 бітам, називається *байтом.*

Таким чином, в одній комірці з 8 двійкових розрядів міститься обсяг інформації в один байт. Саме тому обсяг пам'яті прийнято оцінювати *кількістю байт* (210 байт = 1024 байт = 1 Кб, 210 Кб = 1048576 байт = 1 Мб).

Команда розміщується в комбінованій комірці наступним чином (рис. 1.11).

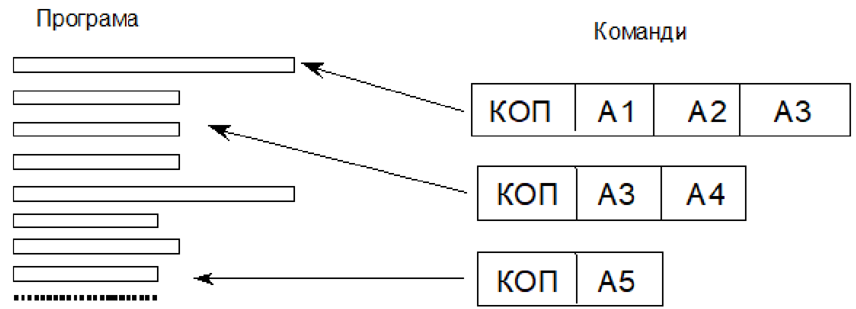



Рисунок 1.11 – Програма в машинних кодах

Перший байт містить код операції (КОП) (наприклад, +, –, *), яку необхідно виконати над вмістом комірок пам'яті. В одній, двох або трьох комірках (операндах команди) по 2 або по 4 байти містяться адреси комірок (A1, A2, A3), над якими потрібно виконати зазначену операцію. *Номер першого байта команди* називається її *адресою*. Послідовність цих команд називається *програмою в машинних кодах*.

Для розміщення даних в комірках проводиться їх запис за допомогою кодування через нулі та одиниці. При кодуванні кожен символ, що вводиться з клавіатури, замінюється послідовністю *8 двійкових розрядів* відповідно до стандартної кодової таблиці, тобто один символ займає один байт. Наприклад, відповідно до таблиці кодів ASCII D → 01000100; F → 00100110; 4 → 00110100; ? → 0011110.

При кодуванні числа перетворюються на двійкове представлення. Наприклад,

2 = 1·21 + 0·20 = 102;

5 = 1·22 + 0·21 + 1·20 = 1012;

256 = 1·28 = 1000000002.

При роботі з числами розрізняють:

1) цілі: ±n;
2) дійсні:
   - з фіксованою десятковою точкою: ±n.m;
   - з плаваючою десятковою точкою (експоненційна форма): ±n.mE±p, де n, m – ціла та дробова частини числа, p – порядок; ±0.xxxE±p – нормалізований вигляд.

Із збільшенням числа кількість розрядів для його представлення в двійковій системі різко зростає, тому для розміщення великої кількості виділяється кілька *розміщених поспіль однобайтних комірок*. У цьому випадку *адресою такої розширеної комірки є адреса першого байта*. Один біт такої комірки виділяється під знак числа. Числа, розміщені в такий спосіб – цілі.

Для зберігання дійсних чисел їх попередньо призводять до нормалізованого виду. Наприклад, 35,6 = 0.356·10+2, де 0.356 – мантиса, +2 – порядок. Після цього переводять порядок та мантису у двійкову систему. Таке число запам'ятовується в комбінованій комірці, один байт якої містить порядок, інші – містять мантису. Числа, розміщені в такий спосіб – дійсні.

Для перетворення команд мови програмування на машинну мову використовується програма. Цей процес перетворення називається трансляцією (від англ. Translation – переклад).

*Транслятор* – це програма, яка здійснює переклад текстів з однієї мови на іншу, тобто транслятор «перекладає» програму з вхідної мови системи програмування на машину мову ПЕОМ, на якій функціонує дана система або функціонуватиме програма, що розробляється; або на проміжну мову програмування, яка вже реалізована або підлягає реалізації.

Одним із різновидів транслятора є *компілятор*, що забезпечує переклад програм з мови високого рівня (наближеного до людини) на мову нижчого рівня (близька до ЕОМ), або машинозалежна мова. Таким чином *компіляція* (від англ. compile – збирати) – це спосіб *трансляції*, при якому здійснюється переклад усього тексту програми, збір перед її виконанням та запис у пам'ять комп'ютера. Компілятор виділяє місце в пам'яті для кожної змінної.

Інший спосіб трансляції – це *інтерпретація*.

*Інтерпретація* (від англ. interpretation) – спосіб трансляції, при якому кожна інструкція програми перекладається в машинні коди та виконується, і тільки після виконання одного фрагмента програми процесор переходить до обробки іншого фрагмента.

Результат обробки вихідного модуля компілятором – *об'єктний модуль* (object code, у мові С має розширення *.obj). Він не може бути виконаний, тобто це незавершений варіант машинної програми, тому що, наприклад, до нього повинні бути приєднані модулі стандартних бібліотек. Тут компілятор (compiler) – це вид транслятора, що представляє програму-перекладач вихідного модуля на мову машинних команд.

*Виконуваний* (*абсолютний, завантажувальний*) *модуль* створює інша спеціальна програма – *компонувальник*. Її ще називають редактором зв'язків (Linker). Вона створює модуль, придатний для виконання на основі одного або декількох об'єктних модулів.

*Завантажувальний модуль* (Load module, розширення *.exe) – це програмний модуль, представлений у формі, яка придатна для завантаження його в пам'ять та виконання.

Помилки, які допускаються при написанні програм, поділяють на синтаксичні та логічні.

*Синтаксичні помилки* – це порушення формальних правил написання програми конкретною мовою. Вони виявляються на етапі трансляції і можуть бути легко виправлені.

*Логічні помилки* поділяють на *помилки алгоритму* та *семантичні помилки*. Такі помилки можуть бути знайдені та виправлені лише розробником програми.

Причина *помилки алгоритму* – невідповідність побудованого алгоритму ходу отримання кінцевого результату сформульованої задачі.

Причина *семантичної помилки* – неправильне розуміння сенсу (семантики) операторів мови.

Компілятори, бібліотеки та інструменти розробки мовою С є практично для всіх систем. Програми мовою С відрізняються переносимістю між платформами лише на рівні вихідного коду.

Мова С сильно вплинула на індустрію розробки програмного забезпечення. З одного боку, синтаксис багатьох його інструкцій є основою таких мов, як С++, С#, Java, PHP. З іншого – він використовується в якості посереднього у деяких системах програмування, коли програма спочатку транслюється в програму мовою С, і тільки потім компілюється компілятором мови С для отримання остаточного виконавчого модуля.

Мову С називають комп'ютерною мовою середнього рівня. Але це не означає, що вона менш досконала у порівнянні з традиційними мовами високого рівня. В ній закладено можливості розробки конструкцій, притаманних мовам високого рівня, вона дозволяє маніпулювати бітами, байтами і адресами, тобто тими базовими елементами, з якими працює комп'ютер.

## 1.7 Теорія структурного програмування

З появою ЕОМ третього покоління виник новий підхід до розробки алгоритмів і програм, який отримав назву *структурного проєктування програм*.

Переваги структурного програмування в порівнянні з інтуїтивним неструктурним програмуванням:

1) зменшення труднощів тестування програм;
2) більш висока продуктивність програмістів;

3) ясність і чіткість програм, що спрощує їх супровід;
4) ефективність програм.

Для більшості завдань алгоритми їх вирішення є досить великими і громіздкими. При програмуванні потрібно намагатися отримати високоефективну програму, яку можна легко читати і легко модифікувати. Саме з цією метою виконують *декомпозицію* складного алгоритму поставленої задачі, тобто здійснюють його розбиття на окремі більш прості підзадачі, потім роблять декомпозицію підзадач і так далі. Для цього застосовують прийоми *структурного програмування.*

*Структурне програмування* – методологія розробки програмного забезпечення, в основі якої лежить представлення програми у вигляді ієрархічної структури блоків.

В основу структурного програмування покладена вимога, щоб кожен модуль алгоритму (програми) проєктувався з єдиним входом і єдиним виходом. Програма представляється у вигляді множини вкладених модулів, кожен з яких має один вхід і один вихід.

Структурне програмування засноване на проєктуванні зверху-вниз (*спадне проєктування*). Ідеї спадного проєктування сформульовані в монографіях Дж. Хьюза, Э. Йодана [3, 4] та С. Гурмана [5].

Метод спадного проєктування – найбільш природний підхід до проєктування систем. Основна ідея його полягає в тім, що задачу спочатку розглядають у цілому і складають укрупнену схему рішення. Потім кожну операцію, починаючи із самої верхньої, необхідно розглядати детально, не вдаючись у деталі операцій, що знаходяться на більш низьких рівнях. Потім деталізують операції більш низького рівня і т.д.

Таким чином, спадне проєктування містить у собі розбивку великої задачі на менші (декомпозиція задачі), тобто підзадачі, що повинні розглядатися окремо в порядку ієрархічного зв'язку.

Розбиття алгоритму на окремі функції та/або модулі здійснюється через використання функціональної та/або модульної декомпозиції відповідно.

*Функціональна декомпозиція* – це метод розбиття великої програми на окремі функції, коли загальний алгоритм поділяють на окремі кроки, які потім оформлюють у вигляді окремих функцій.

Реалізація декомпозиції спирається на виконання наступних правил:

– програма розробляється як послідовність більш дрібних дій;

– кожна проведена деталізація докладно описується у вигляді абстрактного оператора, який повинен *однозначно формулювати* необхідну дію;

– всі абстрактні дії замінюються на групи операторів обраної мови програмування.

При цьому треба пам'ятати, що кожна деталізація – це лише один із варіантів рішення, і тому необхідно переконуватися, що:

– вирішення приватних задач призводить до рішення загальної задачі;

– обрана послідовність дій розумна і ефективна;

– побудована декомпозиція дозволяє отримувати команди, які легко реалізуються обраною мовою програмування.

Одиницею компіляції у мові С є окремий файл (модуль).

*Модульна декомпозиція* представляє собою розбиття програми на кілька окремих файлів, кожен з яких вирішує окрему конкретну задачу і, як правило, полегшує процес її роботи. Код програми, поділений на окремі файли, дозволяє частині цього коду використовувати інші програми.

Метод спадного проєктування часто називають методом *покрокового удосконалювання.* Використовуючи цей метод, необхідно шляхом поступового розгляду виконати декомпозицію, тобто складну задачу розбити на більш прості, а потім крок за кроком, шляхом подальшої деталізації розробити логічну схему алгоритмів рішення задачі.

Структурне програмування стало основою всього, що зроблено в методології програмування, включаючи і об'єктне програмування.

Отже, принципи структурного програмування наступні:

*Принцип 1.* Слід відмовитися від використання оператора безумовного переходу goto.

*Принцип 2.* Будь-яка програма будується з трьох базових керуючих конструкцій: послідовність, розгалуження, цикл.

*Принцип 3.* У програмі базові керуючі конструкції можуть бути вкладені одна в одну довільним чином.

*Принцип 4.* Фрагменти програми, що повторюються, можна оформити у вигляді підпрограм (процедур і функцій). Таким же чином (у вигляді

підпрограм) можна оформити логічні цілісні фрагменти програми, навіть якщо вони не повторюються.

*Принцип 5.* Кожну логічно завершену групу інструкцій слід оформити як блок.

*Принцип 6.* Всі керуючі конструкції повинні мати один вхід і один вихід.

*Принцип 7.* Розробка програми ведеться покроково, методом «зверху вниз».

### 1.8 Програмування на основі мови C++

Розглянемо коротко основні частини загальної структури програми.

У мові С++ фундаментальним поняттям є *інструкція* (*операція*, *оператор*, *функція*), яка є описом певного набору дій. Таким чином, програма, написана мовою С++, складається із *послідовності інструкцій*.

У мовах програмування замість номерів коміркам пам'яті прийнято давати *імена* (*ідентифікатори*), а вміст комірок називати *змінними*, чи *константами*, залежно від того, змінюється значення комірки у процесі роботи чи ні.

Визначимо поняття ідентифікатора та поняття ключових слів.

*Ідентифікатор* – це ім'я програмного об'єкта (константи, змінної, мітки, типу, функції, модуля, поля у структурі). В ідентифікаторі можуть використовуватися латинські літери, цифри та знак підкреслення.

Існують деякі обмеження на завдання імені, а саме.

*Першим символом* ідентифікатора може бути буква або знак підкреслення, але не цифра.

*Пробіли всередині ідентифікатора не допускаються*.

Довжина ідентифікатора визначається версією транслятора С++ та редактора зв'язків (*компонувальника*). Сучасна тенденція – зняття обмежень довжини ідентифікатора.

При назві об'єктів слід дотримуватися *загальноприйнятих угод*:

– ідентифікатор змінної зазвичай пишеться малими літерами, наприклад index (для порівняння: Index – це ідентифікатор типу чи функції, а INDEX – константа);

– ідентифікатор повинен нести будь-яке смислове навантаження, пояснюючи призначення об'єкта у програмі, наприклад birth_date (день народження) або sum (сума);

– якщо ідентифікатор складається з декількох слів, як birth_date, то прийнято або розділяти слова символом підкреслення (birth_date), або писати кожне наступне слово з великої літери (birthDate).

Розділювачами ідентифікаторів об'єктів можуть бути:

– пробіли;

– символи табуляції, переходу рядка та сторінки;

– коментарі (грають роль пробілів).

Наявність роздільників не впливає на роботу програми.

У С++ великі та малі літери – це *різні символи*. Ідентифікатори Name, NAME, name – це різні об'єкти.

*Ключові (зарезервовані)* слова не можуть бути використані в якості ідентифікаторів.

Ключові слова С++:

| | | | |
|---|---|---|---|
| auto | break | case | char |
| const | continue | default | do |
| double | else | enum | extern |
| float | for | goto | if |
| int | long | register | return |
| short | signed | sizeof | static |
| struct | switch | typedef | union |
| unsigned | void | volatile | while |

Перед компіляцією програма обробляється спеціальною програмою – *препроцесором*, який працює під управлінням *директив*.

Препроцесорні директиви починаються з символу #, за яким слідує найменування директиви, яке вказує на *поточну операцію препроцесора*.

Препроцесор вирішує ряд задач стосовно попередньої обробки програми. Основною з цих задач є «підключення» до програми *заголовних файлів* (звичайних текстів) з декларацією (об’явою) *стандартних бібліотечних функцій*, які використовуються у програмі. Найменування такої директиви: *include* (підключити), а загальний формат її використання наступний:

#include <ідентифікатор_файлу.h>

де h – розширення заголовних файлів.

Якщо ідентифікатор файлу представлено в кутових дужках (< >), то пошук даного файлу проводиться в *стандартній директорії* з цими файлами. Якщо ідентифікатор файлу представлено в подвійних лапках (” ”), то пошук даного файлу проводиться в *поточній директорії*.

До бібліотек, які найчастіше використовуються, відносяться:

stdio.h – містить стандартні функції файлового введення-виводу;

conio.h – функції для роботи з консоллю (клавіатура, екран монітора);

math.h – математичні функції.

Друге основне призначення препроцесора – це обробка *макровизначень*. Макропідстановка *define* (визначити) має загальний вигляд:

#define <ідентифікатор> <рядок>

Наприклад:

#define PI 3.1416

У ході препроцесорної обробки програми поява у тексті ідентифікатора (ім’я) PI скрізь замінюється значенням 3.1416.

Загальна структура програми мовою С++ має вигляд:

<директиви препроцесора>

<визначення типів користувача – typedef>

<опис прототипів функцій>

<визначення глобальних змінних>

<функції>

У свою чергу функції мають таку структуру:

<клас пам'яті> <тип> <ідентифікатор функції> (<оголошення параметрів>)

{ // початок функції

*код функції*

} // кінець функції

де:

<клас пам'яті> – це характеристика способу розміщення об'єктів у пам'яті (статична/динамічна), яка визначає область видимості та час життя змінної (за замовчуванням – auto);

<тип> – це характеристика інтерпретації даних, тобто сукупність інформації про те, скільки даному об'єкту необхідно виділити пам'яті, який вид має надання інформації та які дії допустимі з нею (за замовчуванням – int).

Клас пам'яті і тип – атрибути необов'язкові і можуть бути відсутні. Тоді значення встановлюються за замовчуванням.

Будь-яка програма, що написана на С++, складається з однієї або більш функцій. Цим функціям, можна давати які завгодно імена. Серед них є одне особливе – main(). Виконання С++-програми починається з функції main().

Наприклад, рядок програми у вигляді:

int main (void)

повідомляє системі, що ім'ям програми є *main()* – головна функція, і що вона повертає ціле число (на це вказує абревіатура int). Ім'я *main()* – це спеціальне ім'я, яке вказує, *де програма має розпочати виконання*. Наявність круглих дужок після слова *main* свідчить про те, що це ім'я *функції.* Якщо вміст цих дужок відсутній або там представлено службове слово void, тоді це означає, що у функцію *main()* не передається жодних аргументів. Тіло функції *main()* обмежене парою фігурних дужок. Усі ствердження програми, які поміщені в цих фігурних дужках, будуть відноситися до цієї функції.

Отже, функції мають *тіло*, яке складається з *даних* і послідовності *операторів* (*команд*). Початок і кінець тіла функції обрамляють фігурними дужками {}.

В С++ кожний *оператор закінчується крапкою з комою* (;).

Програма може містити *коментарі* – це примітки, які допомагають зрозуміти значення (смисл) програми. Вони орієнтовані на читача і ігноруються компілятором. Для коментарів використовується пара символів /* і * /. В С++ було введено варіант коментаря, який починається символами // і закінчується символом переходу на новий рядок.

### 1.8.1 Алфавіт мови С++

У загальному розумінні такого поняття, як *алфавіт мови*, треба знати, що кожному з множини значень, які визначаються одним байтом (від 0 до 255) у таблиці знакогенератора обчислювальної машини ставиться у відповідність символ.

За системою кодування фірми IBM символи з кодами від 0 до 127, що утворюють першу половину таблиці знакогенератора, побудовані за стандартом ASCII і однакові на всіх IBM-сумісних комп'ютерах.

Друга половина символів (коди 128...255) може відрізнятися на різних комп'ютерах. Зазвичай коди від 128 до 175 та від 224 до 239 використовуються для розміщення символів *національного алфавіту*; коди з 176 до 223 відводяться під *символи псевдографіки*; коди з 240 до 255 – під *спеціальні знаки.*

Алфавіт мови С++ включає:

– великі та малі літери латинського алфавіту, а також знак підкреслення;

– арабські цифри від 0 до 9;

– спеціальні символи:

– +(плюс) –(мінус) *(зірочка) /(дрібна риса) =(рівно) >(більше) <(менше) ;(точка з комою) &(амперсант) [ ](квадратні дужки) { }(фігурні дужки) ) () (круглі дужки) _(знак підкреслення) (пробіл) .(крапка) ,(кома) :(двокрапка) #(номер) %(процент) ~(порозрядне заперечення) ?(знак питання) !(кличний знак) \(зворотний слеш);

– розділові символи: пробіл, символи табуляції, переходу рядка, повернення каретки, нова сторінка та новий рядок.

### 1.8.2 Прості типи даних. Оголошення змінних

Програми оперують з різними даними. Деякі дані встановлюються рівними конкретним значенням ще до того, як програма почне виконуватися, а після її запуску зберігають свої значення незмінними на всьому протязі роботи програми. Це *константи*.

Інші дані, які можуть змінюватися, або їм можуть бути привласнені значення під час виконання програми, називаються *змінними*.

Змінним та іменованим константам задаються імена (ідентифікатори).

Крім розходження між змінними і константами існує ще розходження між типами даних. Деякі дані в програмі є цілими чи дійсними числами, деякі буквами чи символами. Комп'ютер повинний мати можливість ідентифікувати й обробляти необхідним образом данні кожного з цих типів.

Дані можуть бути простими і структурованими. Прості дані – це цілі і дійсні числа, текст і покажчики (адреси). Дані структурованого типу – це масиви, записи і файли. Їх ми розглянемо пізніше.

Специфікатори типів, а також розмір пам'яті й область припустимих значень цих типів наведені у таблиці. 1.2.

Таблиця 1.2 – Прості типи даних

| Тип | Специфікатор типу | Представлення в пам'яті | Область значень величини |
|---|---|---|---|
| Символьний | char | 1 байт | -128 до 127 |
| Символьний | unsigned char | 1 байт | 0 до 255 |
| Цілий | int | Залежить від реалізації | |
| Короткий цілий | short | 2 байти | -32768 до 32767 |
| Довгий цілий | long | 4 байти | -2 147 483 648 до 2 147 483 647 |
| Беззнаковий цілий | unsigned | Залежить від реалізації | |
| Беззнаковий короткий цілий | unsigned short | 2 байти | 0 до 65535 |
| Беззнаковий довгий цілий | unsigned long | 4 байти | 0 до 4 294 967 295 |
| Дійсний | float | 4 байти | $\pm 3{,}4 \cdot 10^{-38}$ до $\pm 3{,}4 \cdot 10^{+38}$ |
| Подвійна точність | double | 8 байт | $\pm 1{,}7 \cdot 10^{-308}$ до $\pm 1{,}7 \cdot 10^{+308}$ |

Відзначимо, що представлення в пам'яті й область значень для типів *int* і *unsigned int* не визначені в мові С. За замовчуванням розмір *int* (зі знаком і без знака) відповідає реальному розміру цілого на даній машині. Наприклад, на 16-ти розрядній машині тип *int* завжди 16 розрядів чи 2 байти. На 32-х розрядній машині тип *int* завжди 32 розряди чи 4 байти. Таким чином, тип *int* еквівалентний типам *short int* чи *long int* у залежності від реалізації.

Усі змінні в програмі повинні бути оголошені (*декларовані*). Тобто необхідно оголосити компілятору про їхню присутність у програмі. Де ж у тексті програми з'являються змінні? Усі змінні *повинні бути декларовані раніш,* ніж будуть використовуватися. При цьому можливі дві форми декларації:

– оголошення, що не призводить до виділення пам'яті;

– оголошення, у якому під об'єкт буде виділено обсяг оперативної пам'яті відповідно до його типу. У цьому разі об'єкт можна одночасно *ініціалізувати*, тобто встановити його початкове значення.

Декларація специфікує тип і містить список з однієї чи декількох змінних цього типу, наприклад:

Синтаксис:

<специфікатор типу> <ідентифікатор >[,<ідентифікатор >...];

де <специфікатор типу> – задає тип даних, що представляються змінною, а <ідентифікатор > – це ім'я змінної. В оголошенні може бути задана більш ніж одна змінна шляхом завдання множинного оголошення, у якому ідентифікатори розділені комами. Всі змінні, задані в одному оголошенні, мають той самий тип.

Приклади:

| | |
|---|---|
| int x; | /* оголошена змінна *x* цілого типу*/ |
| unsigned long a, b | /* оголошені дві змінні: *a* і *b*, що мають тип беззнаковий довгий цілий */ |
| double order; | /*оголошена змінна *order*, якій можуть бути присвоєні величини з плаваючою крапкою */ |

Приклад ініціалізації змінної у своїй декларації:

```
char m = ‘Z‘ ;
int i =0;
float eps = 1.0e-5;
```

До будь-якої змінної у декларації може бути застосований кваліфікатор const для вказівки того, що її значення далі *не буде змінюватися*.

```
const double e = 2.71828182845905;
```

### 1.8.3 Використання констант у програмах

Константи – це об'єкти, які не підлягають використанню у лівій частині оператора привласнення, оскільки константа є величиною, що не адресується, і, хоча вона зберігається в пам'яті ПЕОМ, немає жодного способу дізнатися її адресу. У мові С++ константами є:

- самовизначені арифметичні, символьні та строкові (рядкові) дані;
- ідентифікатори масивів та функцій;
- елементи перерахувань.

*Арифметичні* константи можуть бути *цілого* чи *дійсного* типів.

Константи цілого типу мають формат: ±n (+ зазвичай не встановлюється).

*Десяткові* константи представляють послідовність цифр 0...9, перша з яких має бути 0. Наприклад, 98 і 298 – це звичайні цілі константи, якщо потрібно ввести довгу цілу константу, то вказується ознака *L(l)*, наприклад, 298*L* (298*l*). Для такої константи буде виділено 4 байти. Звичайна ціла константа, яка занадто довга для типу int сприймається як long.

Існує система позначень для восьмеричних і шістнадцятирічних констант.

*Восьмеричні* константи представляють послідовність цифр від 0 до 7, перша з яких має бути 0, наприклад: 020 = 16-десяткове.

*Шістнадцятирічні* константи – це послідовність цифр від 0 до 9 і букв від A до F (a...f), що починається символами 0X (0x), наприклад: 0X1F (0x1f) = 31-десяткове.

Восьмеричні і шістнадцятирічні константи можуть також закінчуватися буквою *L(l)* – long. Наприклад, 020L або 0X20L.

Приклади констант цілого типу:

14592; 13,237; 1013L – десяткові*;*

0477; 05033; 01*l* – восьмеричні*;*

0x123; 0X00ff; 0xb8000*l* – шістнадцятирічні.

Константи *дійсного* типу. Дані константи розміщуються у пам'яті по формату double, тоді як у зовнішньому уявленні можуть мати дві форми:

1) з фіксованою десятковою точкою формат запису: ±n.m, де n,m – відповідно ціла і дробова частини числа;

2) з плаваючою десятковою точкою (експоненційна форма): ±n.mE±p, де n,m – відповідно ціла та дробова частини числа; p – порядок; ±0.xxxE±p – нормалізований вигляд, наприклад, $1,25 \cdot 10^{-8}$ = 0.125E-8.

Приклади констант з фіксованою та плаваючою точками:

1.0

-37.25

170e-10

0.13587e+12.

*Символьні* константи. Представляють собою символ, поміщений в одинарні лапки. Наприклад, 'A', 'x' (займає 1 байт). Тип char → ціле int.

Так само використовуються спеціальні послідовності символів (*управляючі послідовності*). При привласненні символьній змінній управляючі послідовності будуть поміщені в одинарні лапки (апострофи).

Наприклад:

'\n' – новий рядок;

'\t'– горизонтальна табуляція;

'\0' – нульовий символ (порожньо).

Текстові символи безпосередньо вводяться з клавіатури, а спеціальні та управляючі символи надаються у тексті *парами текстових символів*.

Приклади подання символьних констант мови С++:

'К', '7', '$', '\n', '\72'.

*Строкові* константи. Вони представляють послідовність символів коду ASCII, які поміщені в лапки (”). У внутрішньому поданні до строкових констант додається нульовий символ '\0', який позначає кінець *рядка*.

*Рядок* – це масив, що складається з символів. Лапки не є частиною рядка, а служать лише для його обмеження.

Наприклад, внутрішнє уявлення константи "01234\0ABCDEF" можна представити наступним чином:

'0','1','2','3','4','\0','A','B','C','D','E','F',' \0'

Приклади строкових констант:

"Клас системи"

"\n\t Аргументи системи\n"

"Відповідність \"11111\""

У кінець строкової константи компілятор автоматично поміщає нуль-символ. Нуль-символ – це цифра 0, яка на друк не виводиться, а у таблиці коду ASCII має код 0.

Наприклад, рядок " " – порожній рядок (нуль-рядок).

Загальні приклади констант:

1550 – ціла константа;

123456789L – константа типу long;

77541U – беззнакова константа ;

173,75; 1,3e-5 – константи з плаваючою крапкою;

'x'  – символьна константа;

“Прізвище “ – строкова константа;

#define SIZE 500 – іменована константа.

### 1.8.4 Оператори і вирази. Основні операції мови С++

Основними елементами, з яких будується програма мовою С++, є оператори. Більшість же операторів складається з виразів. Вираз являє собою об'єднання операцій і операндів. Операндом називається те, над чим виконується операція.

Приклади виразів:

-7

a+55

z=(a+e)/25

b>100

Указівкою на наявність оператора служить символ «;». Наприклад:

x=40 – це вираз (який може бути частиною більшого виразу)

x=40; – це оператор.

Знак операції – це один або більше символів, що визначають дію над операндами. Усередині знаку операції пробіли не допускаються Операції бувають бінарні (із двома операндами) та унарні (з одним операндом). До бінарних операцій належать арифметичні операції, операції відносин та операції присвоювання.

Унарні операції наведені у таблиці 1.3.

Таблиця 1.3 – Унарні операції

| Операція | Дія |
|---|---|
| – | Змінює знак величини, що стоїть праворуч |
| ++ | Додає одиницю до значення змінної, що стоїть ліворуч (постфіксна форма), чи до змінної, що стоїть праворуч (префіксна форма) |
| – – | Аналогічно ++, але віднімає 1. |
| & | Одержання адреси операнду. |

Постфіксна форма: змінна змінюється після того, як її значення використовується.

Префіксна форма: змінна змінюється перед тим, як її значення використовується.

Арифметичні операції наведені у таблиці 1.4.

Таблиця 1.4 – Арифметичні операції

| Операція | Дія |
|---|---|
| + | Додавання |
| – | Віднімання |
| * | Множення |
| / | Ділення |
| % | Ділення по модулю (дає залишок від ділення). Використовується тільки для цілих чисел |

Порядок виконання операцій:

– вирази в круглих дужках;

– функції (стандартні математичні, функції користувача);

– * / виконуються зліва направо;

– + — зліва направо.

Порядок виконання операцій можна визначати круглими дужками, тоді вираз у дужках виконується передусім (зліва направо).

Операції відносин наведені у таблиці 1.5.

Бінарні операції відносин порівнюють значення ліворуч зі значенням праворуч і виробляють значення **1**(*true*), якщо вираз є істиною, і **0**(*false*), якщо вираз є неправдою. Типом результату є *int*.

Таблиця 1.5 – Операції відносин

| Операція | Відношення |
|---|---|
| < | Перший операнд менше, ніж другий операнд |
| > | Перший операнд більше, ніж другий операнд |
| <= | Перший операнд менше або дорівнює другому операнду |
| >= | Перший операнд більше або дорівнює другому операнду |
| = = | Перший операнд дорівнює другому операнду |

| Операція | Відношення |
|---|---|
| ! = | Перший операнд не дорівнює  другому операнду |

У мові С знак рівності (=) не означає дорівнює. Він означає ***операцію присвоювання*** деякого значення змінної.

**Увага!** Лівим операндом операції присвоювання може бути лише іменована змінна чи змінна, яка опосередковано адресована покажчиком. Розглянемо приклади неприпустимих виразів:

– присвоювання константі: 100 = a + b;

– присвоювання функції: getch() = i;

– присвоювання результату операції: (x+8) = 13+z;

Операції присвоювання в С++ можуть обчислювати і присвоювати значення в одній операції. Використовуючи складені операції присвоювання замість двох окремих операцій, можна скоротити код програми і поліпшити її ефективність (табл. 1.6).

Таблиця 1.6 – Складені операції присвоювання

| Операція | Дія |
|---|---|
| += | Додавання з присвоюванням |
| –= | Віднімання з присвоюванням |
| *= | Множення з присвоюванням |
| /= | Ділення з присвоюванням |
| %= | Залишок від ділення з присвоюванням |

Кожна з цих операцій змінює змінну, що стоїть ліворуч, на величину, що стоїть праворуч. При присвоюванні тип правого операнду перетвориться до типу лівого операнду.

Операція складеного присвоювання складається з простої операції присвоювання, скомбінованої з іншою бінарною операцією. У складеному присвоюванні спочатку виконується операція, специфікована арифметичним оператором, а потім результат присвоюється лівому операнду. Вираз складеного присвоювання, наприклад, має вид:

<*вираз* 1> += <*вираз* 2>

і може бути зрозумілий як:

<*вираз* 1> = <*вираз* 1> + <*вираз* 2>

Наприклад:

int a=7;

a+=10;     те ж саме, що  a=a+10;

Приклади використання складених операцій присвоювання:

```
int n, a, b, c, d;
n = 2; a = b = c = 0;
a = ++n;      //  n=3, a=3
a+ = 2;       //  a=5
b = n++;      //  b=3, n=4
b- = 2;       //  b=1
c = --n;      //  n=3, c=3
c* = 2;       //  c=6
d = n--;      //  d=3, n=2
d+ =4;        //  d=7
d% = 2;       //  d=1
```

## Логічні операції

Перелік логічних операцій у порядку зменшення відносного пріоритету та їх позначення представлено в таблиці 1.7.

Таблиця 1.7 – Логічні операції

| Операція | Назва | Дія |
|---|---|---|
| ! | Заперечення (логічне ***НЕ***) | Результат має значення «істина» (1), якщо операнд має значення «неправда» (0), і навпаки. |
| && | Кон'юнкція (логічне ***І***) | Результат операції має значення «істина» (1), якщо обидва операнди мають значення «істина» (1), і «неправда» (0) у противному випадку. Якщо один з операндів має значення «неправда», то результат також має значення «неправда». Якщо перший операнд має значення «неправда», то другий операнд не обчислюється. Операндами логічних операцій звичайно є умовні вирази. Наприклад: a>6. |

| Операція | Назва | Дія |
|---|---|---|
| \|\| | Диз'юнкція (логічне ***АБО***) | Результат операції має значення «істина» (1), якщо один чи обидва операнди мають значення «істина», і «неправда» (0) у противному випадку. Якщо один з операндів має значення «істина», то результат також має значення «істина». Якщо перший операнд має значення «істина», то другий операнд не обчислюється. |

Вираз 5>2 && 4>7 має значення «неправда», оскільки тільки перший вираз має значення «істина».

Вираз 5>2 || 4>7 має значення «істина» оскільки один з виразів має значення «істина».

Операція ***«,» (кома)***

Ця операція використовується при організації строго гарантованої послідовності обчислення виразів. Форма запису:

вираз1**,** вираз2**,** ...**,** виразN;

вираз1, вираз2, ..., N обчислюються гарантовано послідовно і результатом операції стає значення виразу N.

Наприклад, m = (i = 1**,** j = i ++**,** k = 6**,** n = i + j + k);

Послідовність обчислень буде наступна:

i = 1, j = i = 1, i = 2, k = 6, n = 2 + 1 + 6, і в результаті m = n = 9.

Даний приклад нічим не відрізняється від такого коду:

i = 1; j = i; i ++; k = 6; n = i + j + k; m = n;

Але, у тому випадку, де за синтаксисом допустима лише одна операція, а необхідно розмістити кілька послідовно виконуваних операцій, і використовується операція «**,**» (наприклад, у операторі ***for***).

Також вона використовується при передачі послідовності обчислень в функцію як параметр, при цьому їх необхідно взяти в дужки.

### 1.8.5 Перетворення типів операндів арифметичних операцій

При виконанні операцій можуть зустрічатися операнди різних типів. У цьому випадку вони перетворюються до загального типу відповідно до невеликого набору правил.

Типи операндів перетворюються в порядку збільшення їх «розміру пам'яті», тобто обсягу пам'яті, необхідного для зберігання їх значень. Тому неявні перетворення завжди йдуть від "менших" об'єктів до «великих». Схема виконання перетворень операндів арифметичних операцій:

short, char $\rightarrow$ int $\rightarrow$ unsigned $\rightarrow$ long $\rightarrow$ double

float $\rightarrow$ double

Тобто діють наступні правила:

- значення типів *char* і *short* завжди перетворюються в *int*;
- якщо будь-який з операндів має тип *double*, то другий перетвориться в *double*;
- якщо один з операндів *long*, то інший перетвориться в *long*.

***Увага***: результатом 1/3 буде «0», щоб уникнути такого роду помилок необхідно явно змінювати тип хоча б одного операнда, наприклад: 1.0 / 3.

Типи *char* і *int* можуть вільно змішуватися в арифметичних виразах. Кожна змінна типу *char* автоматично перетвориться в *int*. Це забезпечує значну гнучкість при проведенні певних перетворень символів.

При присвоєнні значення правої частини перетвориться до типу лівої, який і є типом результату. І тут необхідно бути уважним, так як при некоректному використанні операцій присвоювання можуть виникнути неконтрольовані помилки. Так, при перетворенні *int* в *char* старший байт просто відкидається.

Якщо оголошені: float x; int i; то як x = i; так і i = x; призводять до перетворень. При цьому *float* перетворюється в *int* відкиданням дробової частини.

Тип *double* перетворюється на *float* округленням.

Довге ціле перетворюється в більш коротке ціле і змінні типу char за допомогою відкидання зайвих бітів вищого порядку.

При передачі даних функціям також відбувається перетворення типів: зокрема, *char* стає *int*, а *float* – *double*.

У будь-якому виразі перетворення типів може бути здійснено явно за допомогою операції приведення типу. Для цього достатньо перед будь-яким виразом поставити в дужках ідентифікатор відповідного типу.

Вид записи перетворення типу: (тип) вираз;

Її результат – значення виразу, перетворене до заданого типу представлення.

Операція приведення типу змушує компілятор виконати вказане перетворення, але відповідальність за наслідки покладаються на програміста. Рекомендується використовувати цю операцію у виняткових випадках.

Наприклад:

```
float x;
int n = 6, k = 4;
1) x = (n + k) / 3;            // дробова частина буде відкинута
2) x = (float) (n + k) / 3;    // використання операції приведення типу
// дозволяє уникнути округлення результату ділення цілочисельних операндів.
```

### 1.8.6 Введення-виведення інформації у С++

У стандартному С++ існує два основних шляхи введення-виведення інформації: за допомогою потоків і за допомогою традиційної системи введення-виведення, успадкованої від С [6].

Функції форматного введення – виведення даних: ***scanf()*** і ***printf()***.

Функція **printf()** знаходиться в бібліотеці компілятора мови С++, форматує і виводить свої аргументи на екран (стандартний потік виведення) під керівництвом формату, а також вона може використовуватися для перетворення даних.

Функція **printf()** описується в такий спосіб:

**printf («керуючий рядок», аргумент 1, аргумент 2,…)**

Керуючий рядок містить об'єкти трьох типів:

– звичайні символи, що просто виводяться на екран дисплея (копіюються в стандартний вихідний потік);

– специфікації перетворення (таблиця 1.8), кожна з який викликає вивід на екран значення чергового аргументу з наступного списку;

– керуючі символьні константи (таблиця 1.9).

Таблиця 1.8 – Символи перетворення

| Символ перетворення | Тип виведеної інформації |
|---|---|
| d, i | Десяткове ціле число |
| u | Беззнакове десяткове число |
| f | Десяткове число з плаваючою крапкою |
| c | Один символ |
| s | Рядок символів (символи рядка виводяться доти, поки не зустрінеться ознака кінця рядка (\0) чи не буде виведене число символів задане точністю) |
| e | Дійсне десяткове число в експонентній формі |
| o | Восьмеричне ціле число без знаку |
| x | Шістнадцятирічне ціле число без знаку |

Таблиця 1.9 – Символьні константи

| Символьна константа | Значення |
|---|---|
| \ a | Сигнал дзвінок |
| \ b | Повернення на крок |
| \ r | Повернення каретки |
| \ n | Новий рядок |
| \ t | Горизонтальна табуляція |
| \ v | Вертикальна табуляція |
| \ \ | Зворотна похила риска |
| \" | Подвійні лапки |
| \? | Знак питання |

Кожна специфікація перетворення починається зі знаку % і закінчується деяким символом, що задає перетворення. Між знаком **%** і символом перетворення можуть зустрічатися:

– знак мінус, який указує, що перетворений параметр повинний бути вирівняний ліворуч, у своєму полі;

– число, що задає мінімальну ширину поля. Аргумент буде займати поле принаймні зазначеної ширини. Зайві позиції ліворуч чи праворуч (при лівосторонім вирівнюванні) будуть заповнені пробілами;

– крапка, що відокремлює ширину поля від величини, що встановлює точність;

– число (точність) цифр, що задає максимальне число символів які потрібно вивести, чи ж кількість цифр, які потрібно вивести після десяткової крапки в значеннях типів float чи double, чи мінімальна кількість цифр для цілого. За замовчуванням точність дорівнює 6.

– літера h, якщо ціле число повинно бути short, чи літера *l*, якщо – long.

Наприклад:

```
int m=15;
printf ("%6d",m);          // виведе значення a на ширину поля 6. Зайві
                           // чотири позиції ліворуч заповнить пробілами.
float b=25.8;
printf ("%7.2f",b);        // виведе значення b на ширину поля 7, після коми
                           2 знаки. Зайві дві позиції ліворуч заповнить
                           пробілами.
printf ("%-7.2f",b);       // виведе значення b на ширину поля 7, після коми
                           2 знаки. Зайві дві позиції праворуч заповнить
                           пробілами
int a=75;
printf ("\n%d",a);  /      //виведе 75
printf ("\n%c",a);         //виведе K
printf ("\n%o",a);         //виведе 113
printf ("\n%x",a);         //виведе 4b
```

Завдання: що і як виведеться на екран?

```
printf ("\n%-5d%d%f",a,a,b);
```

Функція **scanf()** забезпечує форматне введення даних з клавіатури. Функція **scanf**() читає символи зі стандартного потоку введення (клавіатури), інтерпретує їх відповідно до специфікації.

Так само як для функції **printf()**, для функції **scanf**() указується керуючий рядок і наступний за нею список аргументів. Основне розходження цих функцій полягає у особливості списку. Функція **printf()** використовує імена

змінних, константи і вираз, у той час як функція **scanf()** – тільки покажчики на змінні. Необхідно пам'ятати наступне:

– якщо необхідно ввести деяке значення і присвоїти його змінної одного з основних типів, то перед ім'ям – змінної потрібно писати символ &;

– якщо необхідно ввести значення строкової змінної використовувати символ & не потрібно.

Функція **scanf()** використовує деякі спеціальні символи (пробіли, символи табуляції і «новий рядок») для розбивки вхідного потоку символів на окремі поля. Вона погоджує послідовність специфікацій перетворення з послідовністю полів, опускаючи згадані спеціальні знаки між ними.

Наприклад:

```
int dey;
scanf (“%d”,&dey);
printf(“Сьогодні %5d травня”,dey);
```

Введення-виведення за допомогою стандартних потоків **cin і cout**

Потік – деяка абстракція, що відображає переміщення даних від джерела до приймача.

Потік ***cout*** відповідає пристрою виводу за замовчанням (екран).

Потік ***cin*** відповідає пристрою вводу за замовчанням (клавіатура). Для їх роботи необхідно підключити стандартну бібліотеку **iostream.h**.

Операція << називається операцією помістити в потік. Операція >> називається операцією взяти з потоку.

Наприклад:

```
char Name[10];              // оголошення рядка з 10 символів
cout <<”Ваше ім’я? ”;       // помістити в стандартний потік виводу
                            // рядок «Ваше ім’я?», що означає:
                            // вивід на екран рядка ”Ваше  ім’я?”
cin  >>Name;                // взяти зі стандартного потоку значення і
                            // записати його у змінну Name, що означає:
                            // увід з клавіатури значення змінної Name
```

Приклад використання у програмі на С++ для введення-виведення стандартних потоків ***cin*** і ***cout.***

```
#include <iostream>
using namespace std;
void  main(void)
{
        int  i, j, k;
        cout << "Hello! \n" ;       // \n – перехід на новий рядок
        cout << "Input  i, j: \n";
        cin  >> i >> j ;
        k = i + j ;
        cout << "Sum  i , j = " << k << endl;
        system ("pause");
}
```

### 1.8.7 Стандартні математичні функції

Математичні функції алгоритмічної мови С++ декларовані в файлах <math.h> і <stdlib.h>. У наступних записах аргументи **x** і **y** мають тип double.

Аргументи тригонометричних функцій повинні бути задані в радіанах (2π радіан = 3600). Більшість математичних функцій, наведених у таблиці 1.10, повертають значення (результат) типу double.

Таблиця 1.10 – Математичні функції у мові С++

| Математична функція | Функція у мові С++ |
|---|---|
| $\sqrt{x}$ | sqrt(x) |
| \|x\|, якщо x – ціле | abs(x) |
| \|x\|, якщо x – неціле | fabs(x) |
| $e^x$ | exp(x) |
| $x^y$ | pow(x,y) |
| ln(x) | log(x) |
| $lg_{10}(x)$ | log10(x) |
| sin(x) | sin(x) |
| cos(x) | cos(x) |
| tg(x) | tan(x) |
| arcsin(x) | asin(x) |

| Математична функція | Функція у мові С++ |
| --- | --- |
| arccos(x) | acos(x) |
| arctg(x) | atan(x) |
| arctg(x / y) | atan2(x) |
| $sh(x)=0.5\ (e^{x}-e^{-x})$ | sinh(x) |
| $ch(x)=0.5\ (e^{x}+e^{-x})$ | cosh(x) |
| tgh(x) | tanh(x) |
| остача від ділення *x* на *y* | fmod(x,y) |
| найменше ціле >=x | ceil(x) |
| найбільше ціле <=x | floor(x) |

Приклад використання математичних функцій реалізує алгоритм прикладу 1.1 (див. п.1.5,рис.1.8, стор.11).

```
#include <iostream>
using namespace std;
void main()
{
    int  z=15;
    double x, B;
    setlocale (0, "");            // Коректне відображення кирилиці
    cout<< "Введіть x:";
    cin >> x;
    B=(z*sqrt(x)+pow(x,2))*exp(double(x));
    cout <<"B="<<B;
    cout<<"\n\n";
}
```

### 1.9 Синтаксис операторів мови С++

Оператори мови С ++ можна розділити на три групи:

- оператори декларації;
- оператори перетворення об'єктів;
- оператори управління процесом виконання алгоритму.

Прикладом операторів декларації, з розглянутих раніше, є оператори оголошення (декларації) змінних.

Програмування процесу перетворення об'єктів програми проводиться за допомогою запису виразів. Вираз включає один або кілька операндів і символів

операцій. Будь-який вираз, що закінчується символом «;» (крапка з комою), є оператором.

Найпростіший вид операторів – оператори-вирази.

До простих операторів також належать:

- оператор присвоювання – виконання операцій присвоювання;
- оператор виклику функції – виконання операції виклику функції;
- порожній оператор «**;**».

Оператори управління процесом виконання алгоритму або керуючі оператори в мові С++ належать до наступних класів:

- оператори переходу (умовного та безумовного);
- оператор вибору альтернатив (перемикач);
- оператори організації циклів;
- оператори передачі управління (переходу).

Кожен з керуючих операторів має конкретну лексичну конструкцію, утворену з ключових слів мови С++, виразів і символів-роздільників: **{},: ()**.

Оператори мови С++ записуються у вільному форматі з використанням роздільників між ключовими словами. Допустима вкладеність операторів. У разі необхідності можна використовувати складений оператор – блок, що складається з будь-якої послідовності операторів, укладених у фігурні дужки { }. Після дужки, що закриває блок, символ «**;**» не ставиться.

## Запитання для самоконтролю

1. Перерахувати та коротко пояснити властивості алгоритмів.
2. Коротко охарактеризувати способи опису алгоритмів.
3. Назвіть основні вимоги стандарту до оформлення схем алгоритмів.
4. Назвіть різновиди структур алгоритмів.
5. Які є типи для дійсних чисел?
6. Як описують константи?
7. Перерахувати арифметичні операції та операції відносин.
8. Коли використовується операція «**,**»?
9. Як виконати введення-виведення інформації у С++?

Тест 1.1. Скласти алгоритм, використовуючи словесний опис, для обчислення чисел $\mathbf{A}_i$ за формою $\mathbf{A_i} = (\mathbf{b}*\mathbf{x_i} + \mathbf{c})$, де $\boldsymbol{i}$=1, 2, ... , 30, $\boldsymbol{b}$=15, $\boldsymbol{c}$=20.

Тест 1.2. Скласти блок-схему алгоритму для наступного завдання: ввести з клавіатури три числа **p**, **x**, **y**. Якщо **p** від'ємне число, то обчислити суму чисел **x** і **y**, якщо **p** додатне число чи нуль, то обчислити різницю чисел **x** і **y**.

Тест 1.3. Скласти блок-схему алгоритму обчислення суми елементів матриці A[mxn].

Тест 1.4. Знайдіть помилки у наступній програмі.

```
main
(
  int b;
  float a; s;
  b=7.5;
  s=a*b;
)
```

Тест 1.5. Що виведуть наступні оператори, якщо a=34,76, b=125?

1) printf (“%-7.1f”, a);
2) printf (“%f”, a);
3) printf (“%4d”, b);
4) printf (“%5.4d”, b);

Відповіді на тести у додатку А.

# 2 АЛГОРИТМИ ТА ПРОГРАМУВАННЯ УМОВНИХ ПРОЦЕСІВ

## *2.1 Оператори переходу (умовного та безумовного)*

Часто в програмі необхідно перейти до якоїсь групи операторів. Цей перехід може бути в залежності від того чи іншого результату (умовний перехід) або без залежності від умови (безумовний перехід).

У мові С++ для умовного переходу призначений оператор ***if – else*** (якщо – інакше) та його модифікації.

Безумовну передачу управління виконує оператор ***goto*** (йти до).

Оператор ***goto*** ми розглядати не будемо, тому що його використання суперечить методології структурного програмування (див. п. 1.6).

## *2.2 **Оператор*** if – else

Оператор **if – else** дозволяє організувати розгалуження в програмі і використовується для вибору одного з двох варіантів дій.

Синтаксис запису оператора:

**if** (<*вираз*>) <*оператор1*>

[**else** <*оператор2*>]

Виконання оператора **if – else**. Тіло оператора **if** виконується селективно (вибірково), у залежності від значення <*вираз*>. Тут <*вираз*>, як правило, є логічний вираз або вираз відношення. Спочатку, якщо потрібно, обчислюється вираз, а потім виконується його перевірка на істинність. Якщо значення виразу істина (не нуль), то виконується <*оператор1*>. В противному випадку, тобто, якщо вираз неправдивий, то виконується <*оператор2*>.

З вищенаведеного синтаксису (дужки **[ ]**) виходить, що в мові С++ є два різновиди умовних операторів: простий и повний. Синтаксис простого оператора:

***if*** (<*вираз*>) <*оператор1*>;

Якщо вираз в дужках не нуль, тобто є істиною, то виконується <*оператор1*>, інакше він ігнорується. Оператор <*оператор1*> – простий або складений (блок).

Приклади виразів:

```
if (i>0)
if (1)
if (i)
if ((i+4) <=2)
```

Приклади запису простого умовного оператора:

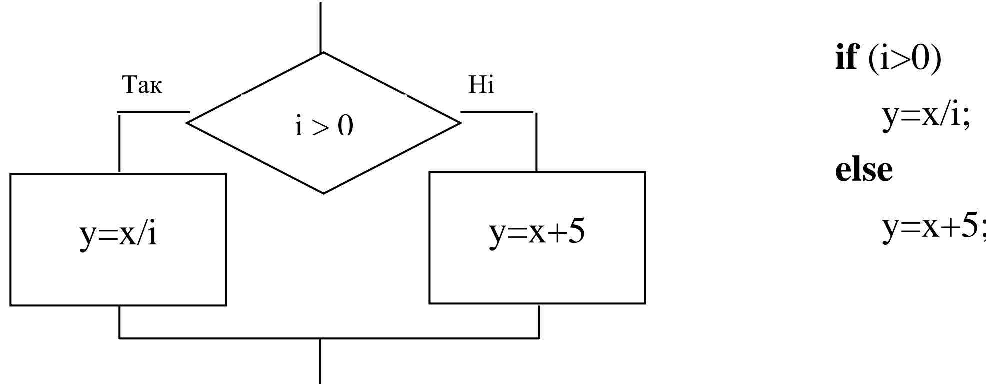
```
if (x>0)  x=0;
if (i!=1) { j++;  s=1; }     // послідовність операцій;
if (i!=1)   j++,  s=1;       // використовується операція «кома»;
if (i) exit(1);              // відповідає запису      if (i!=0) exit(1);
if (1) i=0;                  //відповідає запису       i=0;
if (getch()!=27)             // якщо нажата клавиша, не "Esc".
        k=0;
```

Вираз, що записаний в квадратних дужках, можна опускати. Якщо <*вираз*> неправдивий і опущено ключове слово **else**, то керування передається на виконання оператора, що слідує за оператором **if**. Якщо необхідно виконати не один оператор, у випадку, якщо <*вираз*> є істинним чи неправдивим, то оператори необхідно укласти в дужки {}.

```
if (<вираз>) {
  <оператор 1>
. . .
  <оператор n>
}
[else  {
  <оператор 2>
  <оператор m>
. . .
}]
```

*Приклад 2.1.* Обчислити y=x/i, якщо **i** додатне число, чи y=x+5, якщо **i** негативне чи дорівнює 0.

Нижче наведений фрагмент алгоритму прикладу 2.1 і фрагмент коду програми на С++.




```
if (i>0)
    y=x/i;
else
    y=x+5;
```

У прикладі виконується оператор **y=x/i**, якщо **i** більше нуля. Якщо **i** менше чи дорівнює нулю, то змінної **y** присвоюється значення суми **x** і **5**.

Оператор **if** може бути вкладений у **if** чи **else** іншого оператора **if**. Коли оператори **if** вкладаються, то використовуються фігурні дужки, щоб згрупувати складені оператори, що прояснюють ситуацію. Якщо фігурні дужки відсутні, то компілятор може прийняти неправдиве рішення, поєднуючи кожне **else** з більш близьким **if**, у якого відсутнє **else**.

У прикладі 2.2 **else** асоціюється з внутрішнім оператором **if**. Якщо і менше чи дорівнює 0, то немає значення, що буде присвоєно х.

У прикладі 2.3 фігурні дужки обмежують внутрішній оператор **if** і тим самим роблять слово **else** частиною зовнішнього оператора **if**. Якщо **i** менше чи дорівнює нулю, то значення **i** присвоюється змінній **x**.

Приклад 2.2.

```
if (i>0)
   if (j>i)
       x=j;
   else
       x=i;
```

Приклад 2.3.

```
if (i>0)
{
   if (j>1)
       x=j;
}
else
   x=i;
```

Іноді буває необхідним об'єднати два або більш умовних вирази. Для цього можна використовувати *логічні* операції (див. табл.1.7).

*Приклад 2.4.*

```
int x=5, y=7, z=10;
if (x < y && y < z)
   printf ("x  менше  z \n");
```

У прикладі 2.4 виведеться на екран "x менше z " тому, що обидва вираз x < y та y < z є істиною.

Приклад використання оператора **if – else**, який реалізує алгоритм, наведений у п.1.5 (див. рис. 1.9 стор. 12).

```
#include "iostream"
using namespace std;
void main()
{
  int    x;
  float  y;
  cout<<"x= ";
  cin >>x;
  if (x>=-5 && x<5)
    y=9*pow((double)x,2) + 27*x - 1.25;       //pow() підносить x до ступеня 2
  else
    if (x>=5)
      y=abs( x - 45);                          /*abs() – знаходить модуль числа */
    else
          y= sqrt((double)3*abs(x)/4);         // sqrt() –  обчислює  √
  cout<< "\n y = " << y;
  cout <<endl;
  system ("pause");
}
```

## 2.3 *Оператор вибору альтернатив* switch

Іноді в програмі необхідно зробити вибір одного з декількох варіантів. Це дозволяє зробити оператор вибору альтернатив ***switch*** – ***case*** – ***default***, який ще називають перемикачем.

Синтаксис запису оператора:

```
switch (<вираз>)
{
  case мітка 1 : оператор 1;
  case мітка 2 : оператор 2;
  . . .
  case мітка n : оператор n;
  [default:
     оператор n+1;]
}
```

Виконання оператора. В операторі **switch** обчислюється (якщо потрібно) цілий вираз в дужках (його ще називають селектором), і його значення порівнюється з усіма мітками. При збігу виконується відповідний варіант (один чи кілька операторів). Усі мітки в записі оператора повинні бути різними. Варіант із ключовим словом **default** (інші) реалізується, якщо жоден варіант не виконується. Для припинення наступних перевірок після успішного вибору деякого варіанта використовується оператор **break**, що забезпечує негайний вихід з оператора **switch**.

*Приклад 2.5.* У відповідності до введеної з клавіатури літери вивести назву пори року.

Код на С++, що реалізує алгоритм прикладу 2.5 (рис. 2.1):

```
char lt;
 scanf("%c",&lt);
  switch (lt)
  {
   case 'з': printf("Зима\n"); break;
   case 'в': printf("Весна\n"); break;
   case 'л': printf("Літо\n");  break;
   case 'о': printf("Осінь\n"); break;
   default:
              printf ("Це не пора року");
  }
```

У прикладі 2.5 у тілі оператора **switch** після кожного оператора виводу записаний оператор **break**. Оператор **break** здійснює примусовий вихід з **switch** після виконання одного з цих операторів. Останній оператор **break** не є

обов'язковим, оскільки без нього керування все одно було б передане на кінець оператора **switch**, але він включений для однаковості.

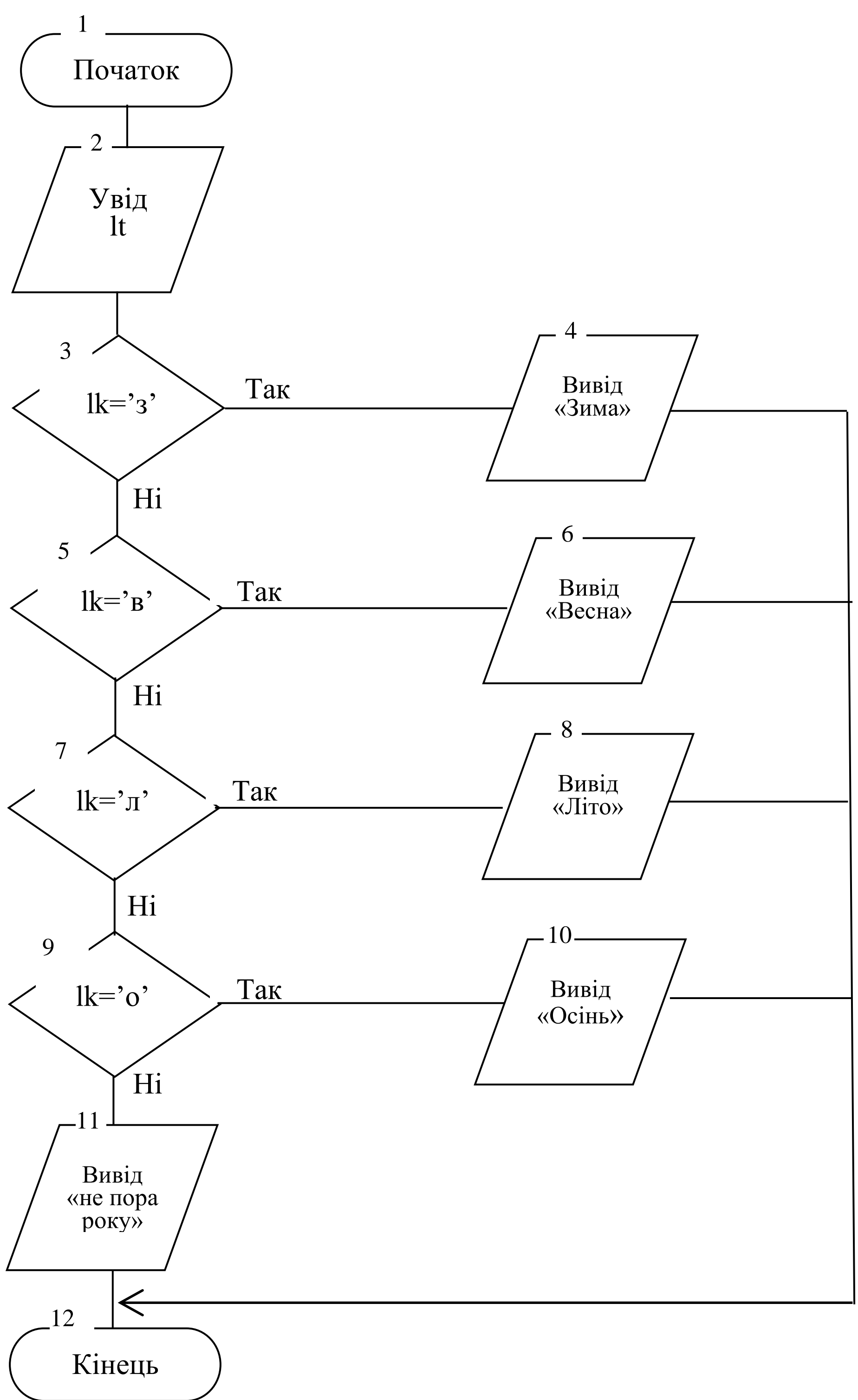



Рисунок 2.1 – Алгоритм рішення прикладу 2.5

Оператор тіла switch може бути позначений множинними мітками, як показано в нижченаведеному прикладі:

```
case 'a':
case 'b':
case 'c':
case 'd':
case 'e':
case 'f': lettera++;
```

Хоча будь-який оператор усередині тіла оператора **switch** може бути поміченим, однак не потрібно оператора, щоб з'явилася мітка. Оператори без міток можуть бути змішані з поміченими операторами. Варто пам'ятати, якщо оператор **switch** передав керування одному з операторів свого тіла, то всі наступні за ним оператори в блоці виконуються, не звертаючи уваги на свої мітки (якщо немає оператора **break**).

У прикладі 2.6 вводиться число у змінну ***k***, далі у операторі **switch** спочатку обчислюється вираз (***k/2***), далі результат обчислення аналізується (тобто порівнюється з мітками), та виводиться напис – парне чи не парне число ***k/2***. Ввід-вивід здійснюється за допомогою стандартних потоків ***cin*** і ***cout***.

*Приклад 2.6.*

```
int k;
cout <<"Введіть  число";
cin>>k;
switch (k/2) {
     case 1:
     case 3:
     case 5: cout <<"\n" <<k/2 <<" – непарне число";  break;
     case 2:
     case 4:
     case 5: cout <<"\n" <<k/2 <<" – парне число";  break;
  }
```

Запитання для самоконтролю

1. Назвіть оператор умовної передачі керування та його різновиди.
2. Як діє оператор умовної передачі керування?

3. Наведіть перелік та призначення логічних операції.

4. Як діє оператор-перемикач switch?

5. Для чого призначений оператора **break**?.

Тест 2.1. Розробити алгоритм і код на С++ для вводу з клавіатури двох цілих чисел. Якщо числа рівні, то вивести напис «Числа рівні».

Тест 2.2. Скласти програму, яка повинна обчислювати Y=2x+7, якщо $5 < x \le 5$, та Z =3k; якщо 0<k <-10, та знаходити найменше з двох чисел (Y чи Z). Значення змінних **x** та **k** уводити з клавіатури.

Тест 2.3. У результаті виконання наступного фрагменту програми на С++

```
int a=5,y=0;
float k=0;
if (a>3)
        y=pow(a,2);
 k=a+25;
```

змінні матимуть значення:

а) a=5 y=25 k=30
б) a=25 y=0 k=30
в) a=25 y=0 k =50
г) a=25 y=25 k=0

Відповіді на тести у додатку А.

# 3 АЛГОРИТМИ ТА ПРОГРАМУВАННЯ ЦИКЛІЧНИХ ПРОЦЕСІВ

У мові С++ існують наступні оператори циклу ***while***, ***for***, ***do – while***

## *3.1 Оператор циклу* while

Синтаксис запису оператора ***while***:
***while*** (вираз)
    оператор 1;

Виконання оператора ***while***. Спочатку перевіряється вираз. Якщо вираз є істиною (не нуль), то виконується оператор, що входить у цикл (тобто оператор 1), а потім вираз перевіряється знову. Ця послідовність дій (перевірка виразу та виконання оператора 1) повторюється до тих пір, поки вираз не стане неправдою. Якщо вираз спочатку неправдивий, то тіло оператора ***while*** (оператор 1) не виконується і керування передається на наступний оператор програми. Оператор ***while*** може також завершитися при виконанні операторів ***break***, ***return*** усередині тіла ***while***.

*Приклад 3.1.* Обчислити суму чисел від 10 до 20.

```
#include "iostream"
using namespace std;
void main()
{
    int s=0;
    int i=10;
    while (i<=20) {
        s+=i;
        i++;
    }
    setlocale (0, "");
    cout<< "Сума чисел від 10 до 20 дорівнює "<<s<<"\n";
}
```

*Приклад 3.2*. Обчислити суму цілих чисел, що вводяться з клавіатури. Числа вводити доти, поки не зустрінеться 0.

```
#include "iostream"
using namespace std;
void main ()
{
    int p=1,  s=0;
    clrscr();
    while (p!=0)
    {
        cout << "Введіть число: ";
        cin >> p);
        s+=p;
    }
  cout << "Сума чисел дорівнює "<<s;
}
```

### *3.2 Оператор циклу* for

Синтаксис запису оператора **for:**

***for*** ([вираз 1];[вираз 2];[вираз 3])
   оператор;

де вираз 1 – вираз ініціалізації;
вираз 2 – умовний вираз;
вираз 3 – вираз модифікації.

В операторі ***for*** використовуються три вирази, що керують роботою циклу. Вони розділені символами «крапка з комою». Вираз ініціалізації (вираз 1) обчислюється тільки один раз до початку виконання оператора чи операторів тіла циклу. Якщо умовний вираз (вираз 2) виявляється істиною (не рівним нулю), тіло циклу (оператор) виконується один раз. Потім обчислюється величина виразу модифікації (вираз 3), і значення умовного виразу визначається знову.

Якщо умовний вираз (вираз 2) є неправдою, то виконання оператора ***for*** закінчується і керування передається наступному оператору в програмі.

Таким чином, тіло оператора ***for*** виконується нуль і більш раз, доти, поки умовний вираз (вираз 2) не стане неправдою.

Кожне з цих трьох виразів може бути відсутнім, але крапку з комою опускати не можна. При відсутності виразу 1 і виразу 3 вважається, що їх просто немає в конструкції циклу, при відсутності виразу 2 покладається, що його значення як би є істиною.

Наприклад наступній цикл

```
for ( ; ; )
{
}
```

є нескінченним циклом, виконання якого переривається за допомогою операторів ***break*** чи ***return***.

Якщо тіло циклу повинне складатися з більш чим одного оператора, то оператори треба взяти у фігурні дужки.

```
for ([вираз 1];[вираз 2];[вираз 3])
{
   оператор 1;
   оператор 2;
    . . .
   оператор n;
}
```

*Приклад 3.3.* Обчислити суму чисел від 10 до 20.

```
#include "iostream"
using namespace std;
void main ()
{
  int i,  s=0;
  for (i=10;i<=20;i++)
     s+=i;
  cout << "Сума чисел від 10 до 20 дорівнює "<< s;
}
```

Можна застосовувати операцію зменшення для модифікуючого виразу (приклад 3.4).

*Приклад 3.4.*

```
for (int n=10; n>0; n--)
```

```
    cout << n <<" секунд!\n";
cout << "Пуск!\n";
```

*Приклад 3.5*. Вивести на екран кожне третє число в діапазоні від 1 до 100. Виводити числа в 5 стовпців.

```
using namespace std;
void main ()
{
    int i, col=0;
    for (i=3;i<=100;i+=3)
    {
        cout << i<< "\t";
        col++;
        if (col==5)
        {
            col=0;
            cout <<"\n";
        }
    }
    cout <<"\n";
}
```

Який цикл обрати: ***while*** чи ***for*** – це справа вподобання. Там, де є проста ініціалізація і покрокове збільшення значення деякої змінної більше підходить цикл ***for***, тому що в цьому циклі організуюча його частина зосереджена на початку циклу. В операторі ***for*** індекс і його граничне значення можуть мінятися усередині циклу.

### *3.3 Оператор циклу* do while

Синтаксис запису оператора ***do while***:

***do***

    оператор;

***while*** (вираз);

Виконання оператора ***do while***. Тіло оператора ***do while*** виконується один чи кілька разів доти, поки вираз стане неправдою. А спочатку виконується

оператор тіла циклу, потім обчислюється вираз. Якщо вираз має значення «неправда», то оператор ***do while*** завершується і керування передається наступному оператору в програмі. Якщо вираз має значення «істина», то тіло оператора виконується знову і знову перевіряється вираз. Виконання тіла оператора продовжується доти, поки вираз не стане рівним «неправда». Оператор ***do while*** може також завершити виконання при виконанні операторів ***break***, ***goto*** чи ***return*** усередині тіла оператора ***do while***.

Приклад 3.6.

```
char b;
do
   cin >> b;
while (b!='*');
```

У прикладі 3.6 спочатку вводиться значення ***b***. Потім перевіряється значення ***b***. Якщо ***b*** містить будь-який символ, але не «*», то тіло оператора виконується знову. Тіло оператора виконується доти, поки користувач не введе «*».

Серед операторів тіла циклу можуть бути оператори:

***continue*** – перехід до наступної ітерації циклу;

***break*** – вихід із циклу.

Запитання для самоконтролю

1. Як діє оператор циклу **while**, **do while**, **for**?
2. Коли використовується оператор **break**?
3. Коли використовується оператор ***continue?***
4. Як організувати вкладені цикли?

Тест 3.1. Скласти програму з оператором **switch**, яка буде з 10 людин рахувати кількість жінок та чоловіків, якщо вводити з клавіатури на запрошення “Введіть стать”, букву **ж** – для жінок, або букву **ч –** для чоловіків.

Тест 3.2. Скласти програму розрахунку суми декількох чисел (кількість чисел увести з клавіатури ) за допомогою операторів:

1) **for**;
2) **while**;
3) **do while**;

Тест 3.3 Обчислити Y= 3x-4 на відрізку [-1,+1] із кроком p=0,2.

Відповіді на тести у додатку А.

# 4 КОРИСТУВАЦЬКІ ФУНКЦІЇ

## *4.1 Призначення функцій*

Принципи програмування мовою С++ засновані на понятті функції. Функція – самостійна одиниця програми, спроєктована для реалізації конкретної задачі. Функції виконують які-небудь дії і можуть одержувати значення величин, використовуваних у програмі. Програми на С++ складаються, принаймні, з однієї функції ***main()***, але можуть містити і більше функцій.

Чому ж користуються функціями? По-перше, вони рятують від повторного програмування. Тобто, якщо конкретну задачу необхідно виконати в програмі кілька разів, ми напишемо відповідну функцію тільки один раз, а потім будемо викликати її завжди, коли це потрібно. По-друге, ми можемо застосовувати одну функцію в різних програмах. Навіть у тому випадку, якщо деяка задача виконується тільки в одній програмі, краще оформити її рішення у виді функції, оскільки функції підвищують рівень модульності програми і, отже, полегшують її читання, внесення змін і коректування помилок. Припустимо, наприклад, ми хочемо написати програму, що робить наступне:

- уводить набір чисел;
- сортує ці числа;
- знаходить їх середнє;
- виводить на друк гістограму.

Відповідну програму можна записати так:

```
void main ()
{
   float list[50];

   read (list);
   sort (list);
   avg (list);
   bargraph (list);
}
```

Зрозуміло, ми повинні запрограмувати чотири функції. Використовуючи змістові імена функції, ми чітко визначаємо, що програма робить і як вона організована.

Після цього можна займатися кожною функцією окремо й удосконалювати її доти, поки вона не буде правильно виконувати необхідну задачу. Додаткова перевага зазначеного підходу полягає в тім, що якщо ми створимо функції досить загального виду, то їх можна буде використовувати в інших програмах.

Що ж потрібно знати про функції? Потрібно знати, як їх можна визначати, як до них звертатися і як установлювати зв'язок між функцією і програмою, яка її викликає.

Порядок створення функції у програмному коді наступний.

**Крок 1.** Оголошення функції у формі *прототипів.* Синтаксис оголошення *прототипу функції*:

[тип значення, що повертається] ім'я функції ([список *формальних параметрів*]);

**Крок 2.** *Визначення функції*. Синтаксис *визначення функції*:

[тип значення, що повертається] ім'я функції ([список *формальних параметрів*])

{

//тіло_функції

…

}

**Крок 3.** *Виклик функції*. Синтаксис *виклику функції*:

ім'я-функції ([список фактичних параметрів]);

*Аргументи функції* – це величини, що передаються функції під час її виклику. Аргументи, що стоять в операторі виклику функції, називаються *фактичними параметрами (аргументами)*. Аргументи, що стоять в операторі визначення та оператора оголошення функції, називаються *формальними параметрами (аргументами)*.

**Увага!** *Тип кожного формального аргументу повинен відповідати типу відповідного фактичного аргументу. Список* (кількість) *формальних аргументів* функції *повинен відповідати списку* (кількості) *фактичних аргументів функції.*

Список параметрів (аргументів) – це список, елементи якого відокремлюються один від одного комами.

Усі параметри функції (що входять до списку параметрів) повинні оголошуватися окремо, причому для кожного з них треба вказувати і тип, і ім'я.

Наприклад:

```
//оголошення функції fun
Int fun(int i, int j, float k); // формальні параметри
….
//виклик функції fun
Fun(345, 78, 90.08); //фактичні параметри
```

При виклику функції формальні параметри набирають значення фактичних параметрів. Якщо функція не має параметрів, то такий порожній список можна вказати в явному вигляді, помістивши для цього в дужки ключове слово void. Функція може повертати значення виклику програму за допомогою оператора return. Тело_функции – це частина визначення функції, обмежена фігурними дужками і розміщена за заголовком функции.

**Увага!** У мові C++ визначення функцій не можуть бути вкладеними, тобто всередині однієї функції не можна оголосити та розписати тіло іншої функції.

Тип функції визначає тип даного, що повертається функцією. Це можуть бути, наприклад, int, float, double і т.д. У разі коли функція нічого не повертає, їй присвоюється тип void. Функція може повертати будь-який тип даних, крім масивів.

## *4.2 Створення і використання простої функції*

*Приклад 4.1.* Написати програму, яка буде виводити на екран титул бланка:

```
***********************************************
УДУНТ
м. Дніпро
пр. Гагаріна, 4
***********************************************
```

Ми бачимо, що повинно двічі виводити рядок із зірочками, отже, можна для її виводу створити функцію, яку назвемо starbar(). Нижче наведений текст програми.

```
#include "iostream"
using namespace std;

void starbar();                    /* оголошення функції starbar() */

void main()
{
   starbar();                      /* виклик функції starbar() */
   cout << "НМетАУ\n";
   cout << "м. Дніпро\n";
   cout << "пр. Гагаріна, 4\n";
   starbar();
}

/* визначення  функції starbar() */
void starbar()
{      /*початок тіла функції */
      int count;
      for (count=1; count <=50; count++)
        cout << "*";
      cout << "\n";
}      /*кінець тіла функції */
```

### *4.3 Оголошення і визначення функцій*

Розглянемо більш детально можливості оголошення функції та її визначення.

Функція повинна бути оголошена *перед її використанням* (*викликом*).

*Оголошення функції* задає тип значення, що повертається функцією, ім'я функції, а також можуть бути специфіковані число і типи аргументів функції, а саме:

[тип, що повертає функція] ім'я функції ([список формальних аргументів]);

*Визначення функції* специфікує ім'я функції, її формальні аргументи (параметри) і тіло функції, тобто оголошення й оператори, що визначають її дії, а саме:

[тип значення, що повертається] ім'я функції ([список формальних аргументів])

{

<тіло функції>

}

У визначенні функції може бути заданий також тип значення, що повертається, та її клас пам'яті.

Специфікатор типу, що стоїть ліворуч від імені функції, визначає тип значення, що повертається функцією. У круглих дужках указується список аргументів (параметрів) – це список (можливо порожній) формальних аргументів, що використовуються функцією. Аргументи використовуються для передачі значень із програми, яка викликає функцію, у функцію. Параметри (аргументи) також повинні бути оголошені. При оголошенні аргументів задають типи та їх імена. Аргументи перелічуються у круглих дужках функції через кому. Далі йде тіло функції, укладене у фігурні дужки.

Тіло функції – це складений оператор, що містить оголошення локальних змінних і оператори. Всі змінні, що оголошені в тілі функції, мають тип пам'яті auto, якщо вони не оголошені інакше. Коли викликається функція, то створюється пам'ять для локальних змінних і виробляється їхня ініціалізація (якщо вона задана). Керування передається першому оператору складеного оператора і починається процес виконання, що продовжується доти, поки не зустрінеться оператор ***return*** для функцій, що повертають значення, чи кінець тіла функції для функцій, що не повертають значення. Керування при цьому повертається у точку виклику.

У прикладі 4.1 функція ***starbar*()** оголошена, як функція, що не повертає значення (void) и не має аргументів (дужки порожні). При визначенні функції її тіло не містить оператор ***return*** тому, що функція не повертає значення.

### *4.4 Аргументи функції. Формальні і фактичні аргументи*

Повернемося до попередньої програми (приклад 4.1). У ній функція *starbar()* виводить 50 зірочок. Змінимо її так, щоб вона виводила різну кількість зірочок, у залежності від значення, що їй передається при її виклику. Тепер нам необхідно, щоб у функцію *starbar()* передавалося число зірочок, що вона повинна вивести, тобто тепер наша функція буде з одним аргументом (приклад 4.2).

Приклад 4.2.

```
#include "iostream"
using namespace std;

void starbar (int limit);            /* оголошення функції starbar() з одним
                                        аргументом limit  цілого типу */

void main()
{
   starbar(25);                          /* виклик функції starbar() */
   cout << "НМетАУ\n";
   cout << "м. Дніпро\n";
   cout << "пр. Гагаріна, 4\n";
   starbar(25);
}

/* визначення  функції starbar() */
void starbar(int limit)
{       /*початок тіла функції */
      int count;
      for (count=1; count <= limit; count++)
        cout << "*";
      cout << "\n";
}       /*кінець тіла функції */ ю
```

Змінна *limit* називається «формальним» параметром (аргументом). Фактично це нова змінна, і в пам'яті комп'ютера для неї повинне бути виділене місце.

Як же користатися цією функцією? Задача в даному випадку полягає в тому, щоб присвоїти деяку величину формальному аргументу *limit***.** Після того як ця змінна одержить своє значення, функція *starbar()* може виконати свою задачу. Змінної *limit* фактично присвоюється значення при виклику функції, а саме:

starbar(25);

Отже 25 – це фактичний аргумент (параметр) і ця величина присвоюється формальному аргументу – змінної *limit*, тобто виклик функції робить наступну дію: *limit* = 25;

Отже, формальний аргумент – змінна, зазначена в описі функції, а фактичний аргумент – конкретне значення, що присвоюється цій змінній програмою, яка її викликала. Тип кожного формального аргументу повинний відповідати типу фактичного аргументу і типу відповідного аргументу в списку типів аргументів функції, якщо такий список мається.

Фактичний аргумент може бути константою, змінною чи навіть більш складним виразом.

Наприклад:

```
int  n =30;
starbar (n);          /*Фактичний аргумент  n є змінною*/
чи
starbar (n/2);        /*Фактичний аргумент n/2  є виразом */
```

Фактичні аргументи можуть бути будь-якою величиною основного, структурного чи адресного типів. Хоча масиви і функції не можуть бути передані як параметри, але покажчики на ці об'єкти можуть передаватися. Всі фактичні аргументи передаються за значенням у викликану функцію. Копія фактичних аргументів присвоюється відповідним формальним параметрам. Функція використовує ці копії, не впливаючи на змінні, з яких копія була зроблена.

### 4.5 Виклики функцій

Виклик функції – це вираз, що передає керування і фактичні аргументи (якщо вони є) функції. Виклик функції має наступне синтаксичне представлення:

ім'я функції ([список фактичних аргументів]);

Ім'я функції обчислюється як адреса функції. Список аргументів, у якому вони їдуть через кому, представляє список фактичних аргументів, що передаються функції. Список аргументів може бути порожнім.

При виконанні виклику функції відбувається заміна формальних аргументів на фактичні. Перший фактичний аргумент відповідає першому формальному аргументу, другий другому і т.д. Викликана функція працює з копією фактичних аргументів, тому будь-яка зміна, зроблена функцією з аргументами, не відобразиться на фактичних величинах, з яких була зроблена копія.

Передача керування здійснюється на перший оператор функції. Виконання оператора ***return*** у тілі функції повертає керування і значення, що стоїть біля нього, в функцію, що її викликала. Якщо оператора ***return*** нема, то керування повертається після виконання останнього оператора тіла функції.

### 4.6 Опис типів функції

Тип функції визначається типом значення, що повертається нею, а не типом аргументів. Функції можуть передавати (повертати) у програму, яка її викликала, одне значення. У таких функціях з'являється тип значення, що повертається, і в тілі функції для повернення значення використовується оператор ***return***.

*Приклад 4.3.* Обчислити і вивести на екран суму двох чисел. Обчислювати суму буде функція *add*(), а виводити результат – функція ***main***().

```
#include "iostream"
using namespace std;
```

```
int add (int x, int y);      /* оголошення функції: функція повертає
                                значення цілого типу, ім'я функції – add, у
                                функцію передаються два аргументи цілого типу
                                */
void main()
{
      int s;
      s = add (5,10);
     cout<< s;
}
/*визначення функції додавання двох чисел */
int add (int x,int y)
{       /* початок тіла функції */
   return (x+y);
}        /* кінець тіла функції */
```

При виконанні оператора ***return*** у функції *add*() керування і значення, що повертається (значення суми **x** та **y** ), передаються у функцію ***main*()**, яка викликала функцію *add()*. У функції ***main()*** це значення присвоюється змінній **s.**

Тіло функції ***main()*** можна записати й в іншому виді:

```
void main()
{
  cout<< add (5,10);
}
```

Звичайне виконання функції не робить ніякого впливу на значення змінних програми, які викликають функції. Щоб мати можливість безпосередньо змінювати значення змінних у програмі, яка викликала функцію, необхідно використовувати аргументи як покажчики.

### *4.7 Покажчики*

***Покажчик*** – змінна, значенням якої є адреса чарунки пам'яті. Тобто покажчик посилається на блок даних з області пам'яті, причому саме на його початок [9, 12].

Покажчик необхідно оголосити перед використанням, як і будь-яку змінну. Оголошення покажчика наступне:

*Тип даних * ім'я покажчика*;

Приклади оголошення покажчиків:

```
int *pi;             /* покажчик  на змінну цілого типу */
char *pc;            /* покажчик  на символьну змінну */
float * pf, *pg;     /*покажчик  на змінні з плаваючою точкою */
```

Принцип оголошення покажчиків такий же, як і принцип оголошення змінних. Відмінність полягає лише в тому, що перед ім'ям ставиться символ зірочки *. Візуально покажчики відрізняються від змінних тільки одним символом. При оголошенні покажчиків компілятор виділяє кілька байт пам'яті, в залежності від типу даних, для зберігання інформації в пам'яті.

Тобто, недостатньо сказати, що деяка змінна є покажчиком. Крім цього, необхідно повідомити ще, на змінну якого типу посилається даний покажчик. Причина полягає в тім, що змінні різних типів займають різне число чарунок, у той час, як для деяких операцій, зв'язаних з покажчиками, потрібно знати обсяг відведеної пам'яті.

Покажчик може посилатися на змінну або функцію. Для цього потрібно знати адресу змінної або функції. Так ось, щоб дізнатися адресу конкретної змінної в С ++ існує унарна операція *взяття адреси* **&**. Така операція витягує адресу оголошених змінних, для того, щоб його присвоїти покажчику.

Щоб отримати значення, що записане в деякій області, на яку посилається покажчик потрібно скористатися операцією *разіменування* покажчика *. Необхідно поставити зірочку перед ім'ям і отримаємо доступ до значення покажчика.

Специфікація типу задає тип змінної, на яку посилається покажчик, а символ зірочка (*) визначає саму змінну як покажчик. Опис виду int *pi; говорить, що pi – це покажчик і що *pi – величина типу int.

```
int  val, i=22;
int  *ptr;
ptr=&i;
val=*ptr;
```

У даному випадку &i означає «покажчик на змінну i». Коли за знаком & записане ім'я змінної, результатом операції є адреса зазначеної змінної. Цю адресу можна записувати в змінні типу «покажчик».

Результатом виконання цього фрагменту є присвоювання значення 22 зміній val.

* – це операція непрямої адресації. Коли за цим знаком іде покажчик на змінну, результатом операції є величина, що поміщається у чарунку із зазначеною адресою.

Приклад коду C++, який буде використовувати покажчики.

```
int var = 123;           // ініціалізація змінної var числом 123
int *ptrvar = &var;      // присвоїти покажчику ptrvar адресу змінної var)
// адреса змінної var витягнута операцією взяття адреси (&)
cout << "&var    = " << &var << endl;
// адреса змінної var, є значенням покажчика ptrvar
cout << "ptrvar  = " << ptrvar << endl;
cout << "var     = " << var << endl;       // значення в змінній var
// вивід значення, що міститься в змінної var через покажчик, операцією
//разіменування покажчика
cout << "*ptrvar = " << *ptrvar << endl;
```

Покажчики використовуються для передачі даних по посиланню, що набагато прискорює процес обробки цих даних (в тому випадку, якщо обсяг даних великий), так як їх не треба копіювати, як при передачі за значенням, тобто, використовуючи ім'я змінної. В основному покажчики використовуються для організації динамічного розподілу пам'яті, наприклад при оголошенні масиву, не треба буде його обмежувати в розмірі. Адже програміст заздалегідь не може знати, якого розміру потрібен масив користувачеві, в такому випадку використовується динамічне виділення пам'яті під масив.

Розглянемо програму, що здійснює обмін значеннями 2-х змінних (приклад 4.4).

*Приклад 4.4.* Обмін значеннями змінних **x** та **y**.

```
#include "iostream"
using namespace std;
```

```
void change (int *u, int *v);      /* u і v є покажчиками */

void main()
{
   int x=5, y=10;
   cout<<"Спочатку x=" <<  x << " і " <<" y= " <<y <<"\n";
   change (&x,&y);
   cout<<"Тепер  x=" <<  x << " і " <<" y= " <<y <<"\n";
}
void change (int *u, int *v)       // Функція обміну значеннями змінних
{
  int temp;
  temp = *u;          /* temp присвоюється значення, на яке вказує u */
  *u = *v;
  *v = temp;
}
```

Спосіб доступу з функції до значень змінних програми, яка викликала функцію, забезпечують покажчики. Оскільки покажчик на змінну містить адресу змінної, то функція може використовувати цю адресу для доступу до значення змінної.

Запитання для самоконтролю

1. Яка структура С++-програми?
2. Назвіть призначення функцій.
3. Як створити функцію?
4. Як оголосити функцію?
5. Які бувають аргументи функцій?
6. Що таке формальні та фактичні аргументи?
7. Як викликати функцію?

Тест 4.1. Написати програму піднесення до квадрату числа від 20 до 100. Піднесення до квадрату повинна виконувати окрема функція з одним аргументом, яка буде повертати значення квадрата аргументу.

Відповіді на тест у додатку А.

# 5 АЛГОРИТМИ І ПРОГРАМИ ОБРОБКИ ЧИСЛОВИХ МАСИВІВ ІНФОРМАЦІЇ

## *5.1 Обробка одновимірних числових масивів інформації*

Наприклад, у програмі необхідно зберігати інформацію про кількість днів кожного місяця. У подібному випадку зручно використовувати масив з 12 елементів, який можна ініціалізувати перед початком роботи програми.

Масив являє собою групу елементів одного типу. Для створення масиву необхідно оголосити тип елементів масиву та їх кількість.

Приклади опису одновимірних масивів наведені нижче.

```
int temp[365];     /* масив з 365 цілих чисел */
float b[20];       /* масив з 20 чисел дійсного типу */
char fio[30];      /* масив з 35 символів */
```

Квадратні дужки (**[ ]**) говорять про те, що **temp**, **b** і **fio** є іменами масивів, а число, укладене в дужки, указує кількість елементів масиву. Окремий елемент масиву визначається за допомогою його номера, який також називається індексом. Нумерація елементів масиву починається з нуля, тому **b[0]** є першим, а **b[19]** останнім, 20-м елементом масиву **b**.

Розмір масиву може задаватися константою або константним виразом. Не можна задавати масив змінного розміру. Для цього існує окремий механізм – динамічне виділення пам'яті.

***Увага***. У мові С++ з метою підвищення швидкодії програми відсутній механізм контролю меж зміни індексів масивів. При необхідності такий механізм повинен бути запрограмований явно. Список початкових значень використовується при необхідності ініціалізувати дані при оголошенні (він може бути й відсутнім).

Звернення до елементу масиву в програмі на мові С++ здійснюється за індексом, наприклад:

```
a [0] = 1;
a [i] ++;
a [3] = a [i] + a [i + 1];
```

Приклад оголошення масиву цілого типу з ініціалізацією значень:

```
int a [5] = {2, 4, 6, 8, 10};
```

Якщо в групі {...} список значень коротше, то елементам, що залишилися, присвоюється 0.

Ініціалізувати можна зовнішні і статичні масиви.

Якщо нічого не засилати в масив перед початком роботи з ним, то зовнішні і статичні масиви ініціалізуються нулем, а автоматичні містять «сміття», що залишилося в цій частині пам'яті.

Нижче наведено приклад ініціалізації масиву кількості днів кожного місяця.

```
#include "iostream"
using namespace std;
int days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
void main()
{
    int index;
    for (index = 0; index < 12; index++)
        cout<< "Місяць " <<index + 1 <<" має " << days [index] << "днів. ";
}
```

Кількість елементів у списку повинне відповідати розміру масиву. А що ж буде, якщо ми помилилися в підрахунку? Наприклад, при ініціалізації вказали список коротше, на два елементи. При цьому в компілятора не виникне ніяких проблем: просто коли він вичерпає весь список з вхідними даними, то стане присвоювати всім іншим елементам масиву нульові значення. У випадку зайво великого списку компілятор видає помилку.

*Приклад 5.1.* Присвоювання парних чисел елементам масиву.

```
#include "iostream"
using namespace std;
void main()  {
  int count, evens[50];
  for (count = 0; count < 50; count++)
    evens[count] = 2*count;
}
```

*Приклад 5.2.* Знайти суму чисел у числовій послідовності *Bn*, розташованих після найбільшого числа послідовності. Значення *n* (кількість чисел у послідовності) і числа послідовності ввести з клавіатури.

Алгоритм для вирішення цього завдання наведений на рисунку 5.1.

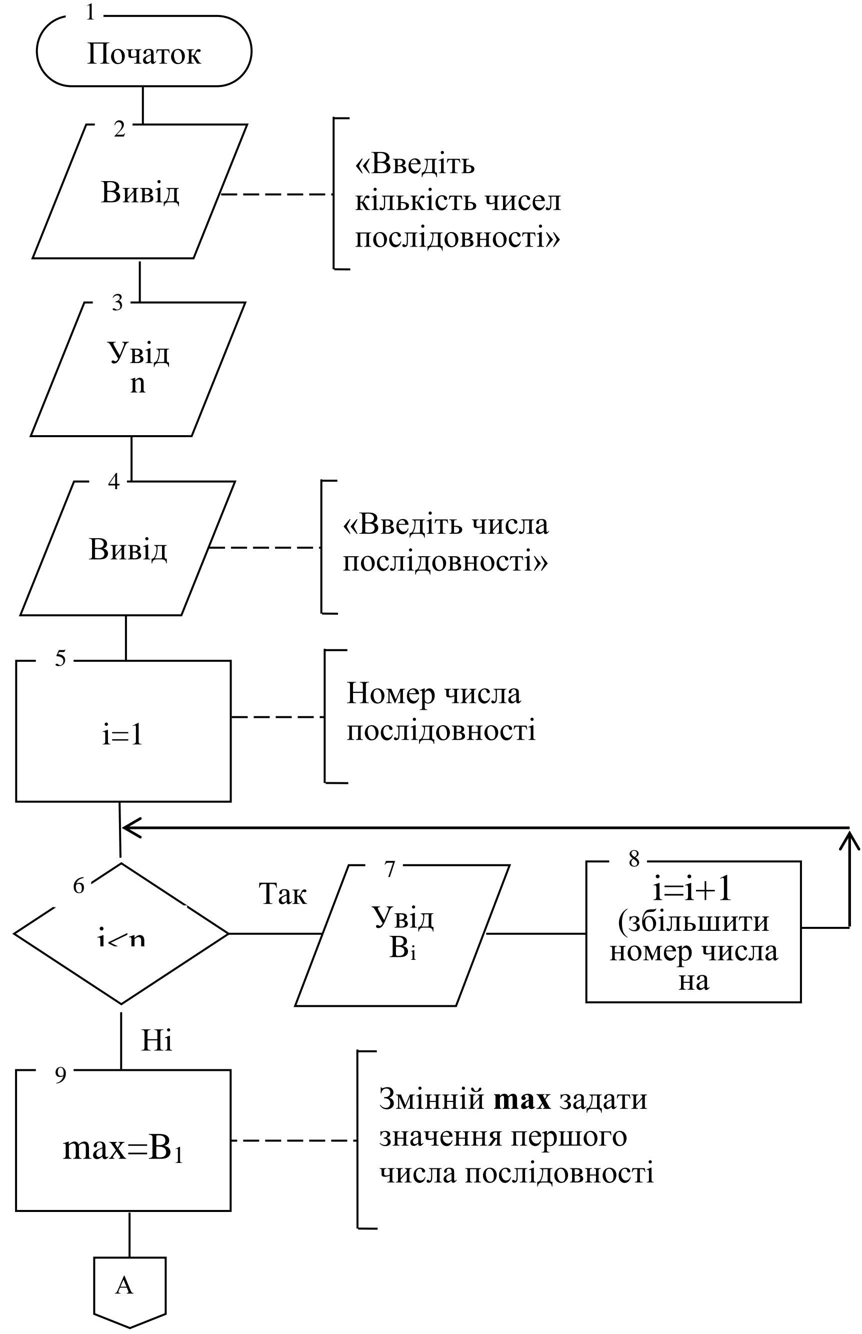
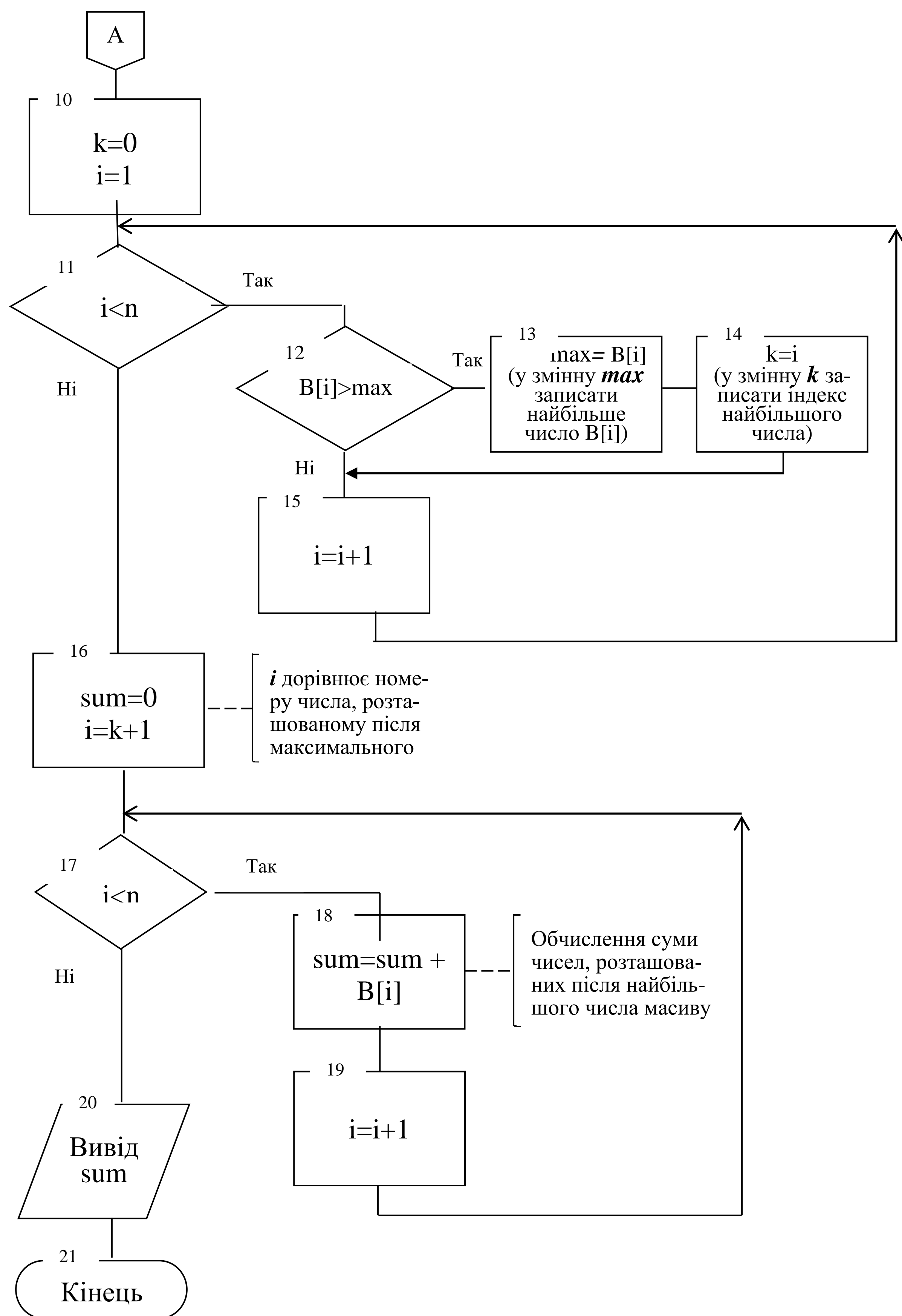



Рисунок 5.1 – Алгоритм для вирішення прикладу 5.2




Продовження рисунка 5.1

Нижче наведений код на C++ прикладу 5.2.

```
#include <windows.h>
#include "iostream"
using namespace std;

void main()
{
	int i, n, max, k, B[20], sum;
	SetConsoleOutputCP (1251);
	cout<<"Введіть кількість чисел послідовності ";
	cin >> n;
	cout<<"Введіть числа ";
	for(i=0; i<n; i++)		// i=0 – перше число масиву має номер 0
		cin >> B[i];
	max=B[0];
	k=0;
	for(i=1; i<n; i++)
	   if(B[i] > max)   // наступне число масиву B[i] більше числа у max
	   {
		max=B[i];
		k=i;		// у змінну k записати індекс найбільшого числа
	    }
	sum=0;
	for(i=k+1; i<n; i++)
		sum+=B[i];
	cout<<"sum=" << sum;
}
```

## 5.2 Обробка багатовимірних числових масивів інформації

Багатовимірний масив чи масив масивів визначається способом завдання списку константних виразів у квадратних дужках, що йдуть за ім'ям масиву. Кожний константний вираз в квадратних дужках визначає число елементів у даному вимірі масиву, так що оголошення двовимірного масиву містить два константних вирази, тривимірного – три і т.д. Наприклад, щоб оголосити масив

з 5 елементів, кожний з яких у свою чергу був би масивом з 10 елементів, варто записати так:

```
int    matr[5][10];  /* оголошення двовимірного масиву, що складає з
                                 5 рядків і 10 стовпців */
```

При зміні другого індексу на одиницю ми пересуваємося уздовж рядка, а при зміні першого індексу на одиницю, пересуваємося вертикально уздовж стовпця [9].

```
float  k[6][7][12];          /* оголошення тривимірного масиву */
```

*Приклад 5.2.* Обчислити середнє значення кожного рядка матриці, що складається з 5 рядків та 7 стовпців.

```
#include "iostream"
using namespace std;
void main()
{
/* ініціалізація двовимірного масиву */
  int  a[5][7] ={ { 2, 1, 4, 8, 12, 0, 3},  { 6, 5, 7, 9, 15, 3, 2},
                  { 3, 0, 2, 5, 37, 1, 9}, { 6, 8, 3, 5, 11, 7, 1},
                  { 5, 0, 4, 2, 1, 13, 12}
              };
  int i, j;
  float s;
  for (i=0; i<5; i++)
  {
   s=0;
   for (j=0; j<7; j++)
      s+= a[i][j];
   cout << "Середнє значення елементів "<< i+1<<" рядка: "<< s/7<<"\ n";
  }
```

Результати виконання програми прикладу 5.2

```
C:\Windows\system32\cmd.exe
Середнє значення елементів 1 рядка: 4.28571
Середнє значення елементів 2 рядка: 6.71429
Середнє значення елементів 3 рядка: 8.14286
Середнє значення елементів 4 рядка: 5.85714
Середнє значення елементів 5 рядка: 5.28571
Для продолжения нажмите любую клавишу . . .
```

## *5.3 Використання покажчиків при роботі з масивами*

Покажчики дозволяють ефективно організувати роботу з масивами. Розглянемо зв'язок між масивом і покажчиком. Робота з масивами тісно пов'язана з використанням покажчиків. Оголошення масиву являє собою сховану форму використання покажчиків.

***Ім'я масиву*** є ***адреса*** області пам'яті, розподіленої під цей масив, або іншими словами – адреса першого елемента масиву. Нехай оголошені наступні дані:

int **a**[5];  – оголошення масиву **a** з 5 елементів цілого типу

int ***q**;  – оголошення покажчика **q** на змінні цілого типу

Нижче наведено символічне зображення оперативної пам'яті, виділеної компілятором для масиву **a[5].**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|

4000 – адреса **a[0]** першого елементу масиву в оперативній пам'яті

Тому що ім'я масиву визначає також його перший елемент, то

a = = &a[0].

Обидві частини рівності визначають адресу першого елемента масиву. Обоє позначення є константами типу покажчик, оскільки вони не змінюються протягом усієї програми.

Після оголошення масиву **a** буде містити 4000. Якщо здійснити наступну операцію присвоювання: **q**=**a**, то **q** також буде містити значення 4000.

Аналогом операції **q**=**a** є **q**=&**a**[0].

Після цього присвоювання:

**q[0]** або ***q** буде позначати **a[0]**;

**q[1]** або *(**q** + **1**) буде позначати **a[1]**;

**q[2]** або *(**q** + **2**) буде позначати **a[2]** і т. д.

І взагалі позначення виду *(**q**+**n**) і **q**[**n**] є повністю еквівалентними. Точно також еквівалентні вирази *(**a**+**i**) і **a**[**i**].

Таким чином, для будь-яких покажчиків можна використовувати дві еквівалентні форми виразів для доступу до елементів масиву: **q[i]** і ***(q + i)**. Перша форма зручніше для читання тексту, друга – ефективніше за швидкодією програми.

На перший погляд здається, що масив і покажчик повністю еквівалентні. Однак є дві суттєві відмінності масиву від покажчика:

- масиву при описі виділяється пам'ять для зберігання всіх його елементів, а покажчику тільки для зберігання адреси;
- адреса масиву назавжди закріплена за ім'ям, тобто ім'я масиву є адресною константою і вираз виду **a** = **q** неприпустимо.

Результат додавання до покажчика або віднімання від нього цілого числа є покажчиком того ж типу, значення якого відрізняється від початкового значення покажчика на число байт, яке визначається добутком цілого числа на розмір даного, яке адресує покажчик. Наприклад, якщо є наступні описи:

int A [20], * pA = A;

double B [20], * pB = B;

то покажчик (**pA + 3**) буде мати значення на 6 байт більше, ніж **pA**, і буде адресувати елемент **A[3]** масиву **A**. Покажчик (**pB + 3**) буде мати значення на 24 байта більше, ніж **pB**, і буде адресувати елемент **B[3]** масиву **B**. З покажчиками типу void подібні операції виконані бути не можуть, оскільки компілятор не знає розміру даного на який адресується. Для покажчиків визначені операції збільшення та зменшення на цілу величину, як альтернативна форма запису виразів:

pA = pA + i;  еквівалентно  pA + = i;

pA = pA - i;  еквівалентно  pA - = i;

pA = pA + 1; еквівалентно  pA ++;   або ++ pA;

pA = pA - 1;  еквівалентно  pA--;  або --pA;

При цьому, робота префіксних і постфіксних операцій ++ і -- збігається з їх роботою для арифметичних даних.

Доступ з функції до масиву забезпечують аргументи покажчики (приклад 5.3).

*Приклад 5.3.* Функція **in()** уводить **M** цілих чисел у масив **a**. Функція **srednee()** обчислює середнє значення **M** чисел масиву **a**.

```
#include "iostream"
using namespace std;
#define M 5
void in (int *b, int n);
float srednee (int *b, int n);
```

```
void main()
{
  int a[M];
  in (a, M);
 cout<<"Середнє значення масиву з "<<M<<" чисел: "<<
         srednee(a, M)<<"\n";
}

void in (int *b,int n)
{
  int i;
  for (i=0;i <n;i++)
     cin >> b[i];
}

float srednee(int *b, int n)
{
   int j;
   long sum=0;

   for (j=0; j <n; j++)
      sum+= *(b+j);
   return float(sum)/n;
}
```

Результати роботи програми прикладу 5.3

```
C:\Windows\system32\cmd.exe
10 5 2 3 8
Середнє значення масиву з 5 чисел: 5.6
Для продолжения нажмите любую клавишу . . .
```

У цьому прикладі **a** – фактичний аргумент функцій **in()** і **srednee()** є покажчиком на перший елемент масиву, що складається з 5 елементів. Таким чином, оператор виклику функції передає їй покажчик. Тому у функціях **in()** та **srednee()** перший аргумент, описаний як покажчик, і він є покажчиком на масив цілих чисел.

Оголошення: int *b та int b[ ] синоніми. Коротше кажучи, коли ім'я масиву передається як аргумент, функції передається покажчик.

Якщо **b** указує на перший елемент масиву, то **b+1**, указує на другий елемент масиву, вираз ***(b+1)** – є значенням другого елемента масиву.

## 5.4 Створення динамічних масивів в С ++

Динамічне виділення пам'яті необхідно для ефективного використання пам'яті комп'ютера. Наприклад, ми написали якусь програму, яка обробляє масив. Під час написання даної програми необхідно було оголосити масив, тобто задати йому фіксований розмір (наприклад, від 0 до 100 елементів). Тоді ця програма буде не універсальною, адже може обробляти масив розміром не більше 100 елементів. А якщо нам знадобляться всього 20 елементів, але в пам'яті виділиться місце під 100 елементів, адже оголошення масиву було статичним, а таке використання пам'яті вкрай неефективне.

В С ++ операції ***new*** і ***delete*** призначені для динамічного розподілу пам'яті комп'ютера. Операція ***new*** виділяє пам'ять з області вільної пам'яті, а операція ***delete*** звільняє виділену пам'ять. Пам'ять, що виділяється, після її використання повинна вивільнятися, тому операції ***new*** і ***delete*** використовуються парами. Навіть якщо не вивільняти пам'ять явно, то вона звільниться ресурсами операційної системи по завершенню роботи програми. Рекомендується все-таки не забувати про операцію ***delete***.

Приклад використання операції ***new***:

int *ptrvalue = ***new*** int;

де *ptrvalue* – покажчик на виділену ділянку пам'яті типу ***int***;

***new*** – операція виділення вільної пам'яті під створюваний об'єкт.

Операція ***new*** створює об'єкт заданого типу, виділяє йому пам'ять і повертає покажчик правильного типу на дану ділянку пам'яті. Якщо пам'ять неможливо виділити, наприклад, в разі відсутності вільних ділянок, то повертається нульовий покажчик, тобто покажчик поверне значення 0. Виділення пам'яті можливо під будь-який тип даних: int, float, double, char і т. д.

Приклад використання операції ***delete***:

***delete*** ptrvalue;

де *ptrvalue* – покажчик на виділену ділянку пам'яті типу ***int***;

***delete*** – операція вивільнення пам'яті.

Вище наведені приклади використання ***new*** і ***delete*** для створення динамічних *змінних*. Найчастіше операції ***new*** і ***delete*** застосовуються для створення динамічних *масивів*.

Оголошення одновимірного динамічного масиву з 10 елементів:

float **ptrarray* = ***new*** float [10];

де *ptrarray* – покажчик на виділену ділянку пам'яті під масив дійсних чисел типу *float*, в квадратних дужках вказуємо розмір масиву.

Після того як динамічний масив став непотрібним, потрібно звільнити ділянку пам'яті, який під нього виділявся. Вивільнення пам'яті, що відводиться під одновимірний динамічний масив, має наступний вигляд:

***delete*** [] *ptrarray*;

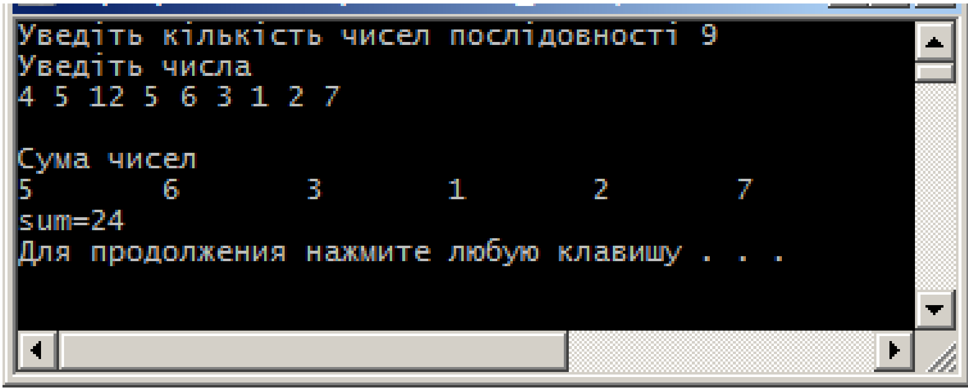
Після оператора ***delete*** ставляться квадратні дужки, які говорять про те, що вивільняється ділянка пам'яті, що відводиться під одновимірний масив.

Нижче наведений другий варіант програми для прикладу 5.2, в якому використовуються покажчики і одновимірний динамічний масив.

```
#include <windows.h>
#include "iostream"
using namespace std;

void main()
{
        int i, n, max, k, sum;
        int *p;
        SetConsoleOutputCP (1251);
        cout<<"Введіть кількість чисел послідовності ";
        cin >> n;

        int *pB = new int [n];      // створення одновимірного динамічного масиву
        p=&pB[0];                   //у покажчик p записати адресу початку масиву
        cout<<"Введіть числа \n";
        for(i=0; i<n; i++)
                cin >> pB[i];
        max=pB[0];
        k=0;
        for(i=0; i<n; i++)
        {
           if(*pB > max)            // число масиву *pB більше числа у змінній max
           {
                max=*pB;
                k=i;                // у змінну k записати індекс найбільшого числа
            }
                pB++;               // перейти до наступного числа масиву
```

```
                                // збільшивши значення покажчика
        }
        sum=0;
        pB=p+(k+1);        //у  покажчик pB записати адресу числа масиву, що
                           // розташоване за найбільшим числом

        cout<<"\nСума чисел" << endl;
        for(i=k+1; i<n; i++)
        {
                cout<<*pB<<"\t";          // вивід числа масиву через операцію
                                          // разйменування покажчика *pB
                sum+=*pB;
                pB++;
        }
        cout<<"\nsum=" << sum << endl;
        delete [] pB;              // вивільнення пам'яті

        system ("pause");
}
```

Результати роботи програми:




У мові С++ можна описати змінну типу «покажчик на покажчик». Це чарунка оперативної пам'яті, в якій буде зберігатися адреса покажчика на якусь змінну. Ознака такого типу даних – повторення символу «*» перед ідентифікатором змінної. Кількість символів «*» визначає рівень вкладеності покажчиків друг в друга.

Відзначимо, що ідентифікатор двомірного масиву – це покажчик на масив покажчиків (змінна типу покажчик на покажчик: int ** m;), тому вираз a[i] [j] еквівалентно виразу * (*(m + i) + j).

Наприклад, двовимірний масив m[3][4] компілятор розглядає як масив трьох покажчиків, кожен з яких вказує на початок масиву зі значеннями розміром по чотири елементи кожен (рис. 5.2).

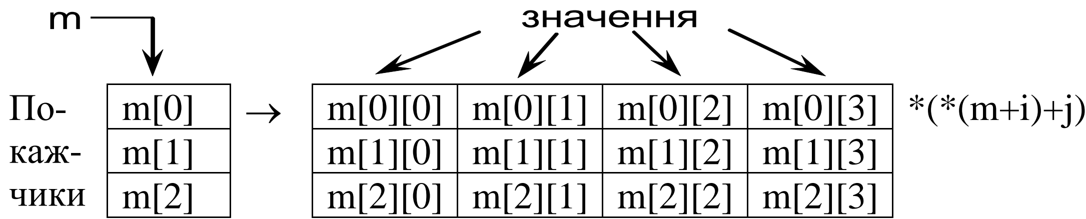



Рисунок 5.2 – Представлення двовимірного масиву m[3][4]

Нижче наведемо приклад, в якому використовується двовимірний динамічний масив.

*Приклад 5.3.* У матриці розміром **n** на **m** знайти такі **k**, що **k**-й рядок матриці збігається з **k**-им стовпцем. Також знайти суму елементів в тих рядках, які містять хоча б один негативний елемент.

Зазначимо, що для виконання цього завдання створимо 4 функції:

sozdanie() – функція, що створює двовимірний динамічний масив;

initialize() – функція, що заповнює матрицю випадковими числами і виводить на екран;

compare() – функція, що порівнює стовпці і рядки;

summa() – функція, що розраховує суму елементів у рядку з негативним елементом.

Перелічені функції оголошені до main().

```
#include <conio.h>
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int** sozdanie(int n, int m);
void initialize( int** A, int n, int m);  // будемо передавати динамічний масив і
                                          // його розмірність в якості параметрів
void compare(int** A, int n, int m);
void summa (int** A, int n, int m);
int main()
{
	setlocale(0, "Russian");
```

```
        int n(0), m(0);
        cout << " Введить кількість рядків у матриці" << endl;
        cin >> n;
        cout << " \n Введить кількість стовпців у матриці " << endl;
        cin >> m;
        cout << endl;
        int **A;
        A = sozdanie(n,m);          // створення двовимірного динамічного масиву
        initialize(A, n, m);        // заповнення матриці
        compare(A, n, m);           // порівняння стовпця і рядка
        summa(A, n, m);             // розрахунок суми чисел рядка з негативним
                                    // числом
        for ( int i = 0; i < n; i++ )
        {
                delete [] A[ i ];
        }
        delete [] A;                // Звільнення пам'яті
       _getch();
        return 0;
}
int** sozdanie(int n, int m)
{
   int **A;                         // Створюємо матрицю введеної розмірності
    A = new int * [n];              // через масив покажчиків
for (int i = 0; i < n; i++) {
      A[i] = new int [m];
   }
   return A;
}

void initialize(int** A, int n, int m)              // Функція заповнення матриці
{
   srand(time(0));
   for(int i = 0; i < n; i++ )
        {
           for(int j = 0;  j < m;  j++ )
                {
                   A[i][j] = rand()%11 - 5 ;
```

```
                        cout << A[i][j] << "     ";
                    }
    cout << endl;
          }
}
void compare(int** A, int n, int m)               // Функція порівняння
{
          int count(0), k(0);
          for(int i = 0; i < n; i++ )   {
                    for(int j = 0;  j < m;  j++ )
                    {
                              if(A[i][j] == A[j][i])        {
                                        count++;
                              }
                              if (count == n)       {
                                        cout << " \n " << i + 1 << " строка збігається з " << i + 1
<< " стовпцем " << endl;
                                        k ++;
                              }

                    }
                    count = 0;
          }
          if(k == 0)
                    cout << "\n\n Немає співпадаючих рядків і стовпців " << endl;
}

void summa (int** A, int n, int m)                // Функція суми
{
          int result(0);
          for(int i = 0; i < n; i++ )
          {
                    for(int j = 0;  j < m;  j++ )
                    {
                              if(A[i][j] < 0)
                              {
                                        for(int count = 0; count < m; count ++)
                                        {
                                                  result += A[i][count];
```

```
                    }
                    cout << "\n Сума елементів  " << i + 1  << " рядка 
дорівнює "<< result << endl;
                    j = m;
                    result = 0;
               }
          }
     }
}
```

У функції створення динамічного масиву sozdanie() спочатку слід створити одновимірний масив покажчиків, потім в циклі в кожен покажчик ми записуємо ще одновимірний масив, таким чином виходить масив масивів, або, двовимірний масив в якому **n** рядків і **m** стовпців.

Друга функція initialize() заповнює масив числами з використанням генератора випадкових чисел і виводить матрицю на екран. В якості параметра цієї функції передається масив, кількість строк і кількість стовпців матриці.

Запитання для самоконтролю

1. Дайте визначення масиву.
2. Як оголосити одновимірний масив? Приведіть приклад.
3. Яке значення має елемент days [1]?

   int days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
4. Як оголосити двовимірний масив? Приведіть приклад.
5. Які індекси має перший елемент двовимірного масиву?
6. Як створити одновимірний динамічний масив?
7. Призначення операцій ***new*** і ***delete***.
8. Як у С++ заповнити масив випадковими числами?

Тест 5.1. Напишіть програму, що забезпечить введення даних у двовимірний масив розміром **m** строк та **n** стовпців, значення яких повинні вводитись з клавіатури. Вивести на екран дані двовимірного масиву у вигляді матриці.

Відповідь на тест у додатку А.

# 6 АЛГОРИТМИ ВПОРЯДКУВАННЯ ІНФОРМАЦІЇ

## *6.1 Упорядкування даних методом вибору*

Розглянемо деякий масив елементів A={$a_1$, $a_2$, $a_3$, ... , $a_n$}. Сутність методу вибору для впорядкування елементів масиву за збільшенням полягає в наступному. При першому перегляді в цьому масиві знаходиться найбільший елемент, який потім міняється місцями з останнім елементом. Таким чином, при першому перегляді на останнє місце попадає найбільший елемент масиву й у наступних переглядах він брати участь не повинен, тому що він вже стоїть на своєму місці (впорядкування за збільшенням) [8].

При наступному перегляді вже аналізуються елементи без останнього, тобто на один елемент менше (n-1), і серед них також знаходиться найбільший елемент, який міняється місцями з (n-1)-им елементом і в подальших переглядах також не бере участь. Упорядкування даних методом вибору завершується, коли значення n буде дорівнювати 1. Блок-схема алгоритму наведена на рисунку 6.1.

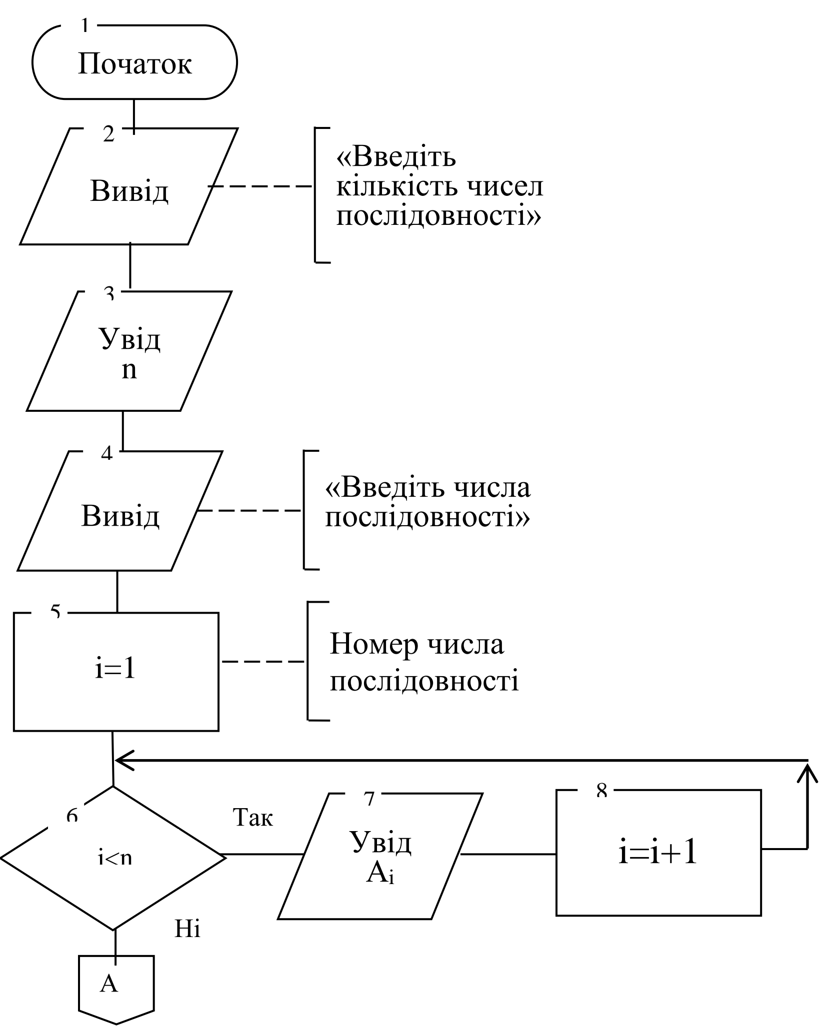
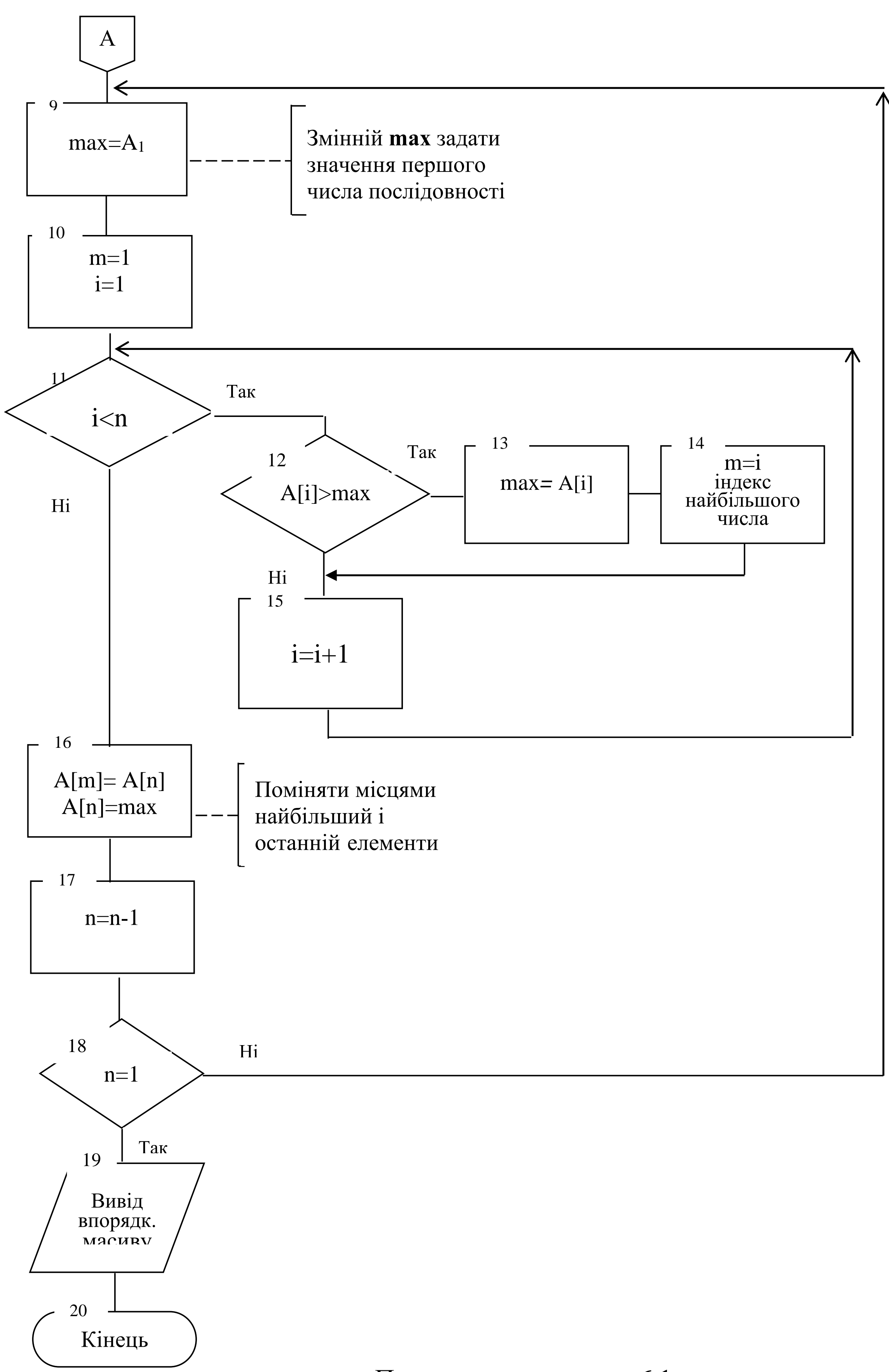



Рисунок 6.1 – Блок-схема впорядкування даних методом вибору




Продовження рисунка 6.1

### 6.2 Упорядкування даних методом попарної перестановки

Метод попарної перестановки (обмінне сортування, метод «бульбашки») полягає в тому, що в масиві елементів $a=\{a_1, a_2, a_3, ..., a_n\}$ порівняються попарно сусідні елементи $a_i$ і $a_{i+1}$. Припустимо, що елементи масиву *a* необхідно встановити в порядку зростання. Тоді, у випадку, якщо $a_i>a_{i+1}$, виконується їхня перестановка. Індекс елементів масиву (i) змінюється від 1 до n-1. У результаті першого перегляду максимальний елемент масиву автоматично попадає на останнє місце, і в подальших переглядах він не буде брати участь [8].

Таким чином, після кожного перегляду кількість елементів, що сортуються, зменшується на 1 і аналізується лічильник перестановок. Процес сортування завершується, якщо на якому-небудь перегляді немає жодної перестановки. Блок-схема методу попарної перестановки представлена на рисунку 6.2.

### 6.3 Упорядкування даних методом Шелла

Сортування методом Шелла [11] подібне обмінному сортуванню (методу попарної перестановки) в тому сенсі, що воно переміщує величини шляхом перестановки пар. Однак воно починає роботу з порівняння величин, що знаходяться на відстані **d** одна від одної. Це призводить до того, що величини, що знаходяться не на місці, будуть переміщатися швидше, ніж при простому обмінному сортуванні. Перша відстань **d** визначається за формулою (6.1):

$$d_1 = \left[ \frac{n+1}{2} \right] \tag{6.1}$$

де n – кількість елементів масиву.

Кожне наступне d визначається як (6.2):

$$d_i = \left[ \frac{d_{i-1}+1}{2} \right] \tag{6.2}$$

На першому перегляді масиву при сортуванні методом Шелла кожна величина порівнюється з величиною, розташованою на **d** позицій далі. Якщо $a_i > a_i + d$, то відбувається перестановка цих елементів. Сортування триває шляхом порівняння наступної величини $a_i + 1$ з величиною розташованою на **d** позицій далі. Так триває до тих пір, поки i + d не стане більше n (тобто поки для наступного елемента знаходиться пара на відстані d). Аналізується лічильник

перестановок: якщо перестановки були, то починається новий перегляд масиву при заданому d; якщо перестановки були відсутні, то процес поновлюється з новим значенням d. Процес сортування завершується, коли порівняння при d = 1 перестають призводити до перестановок.

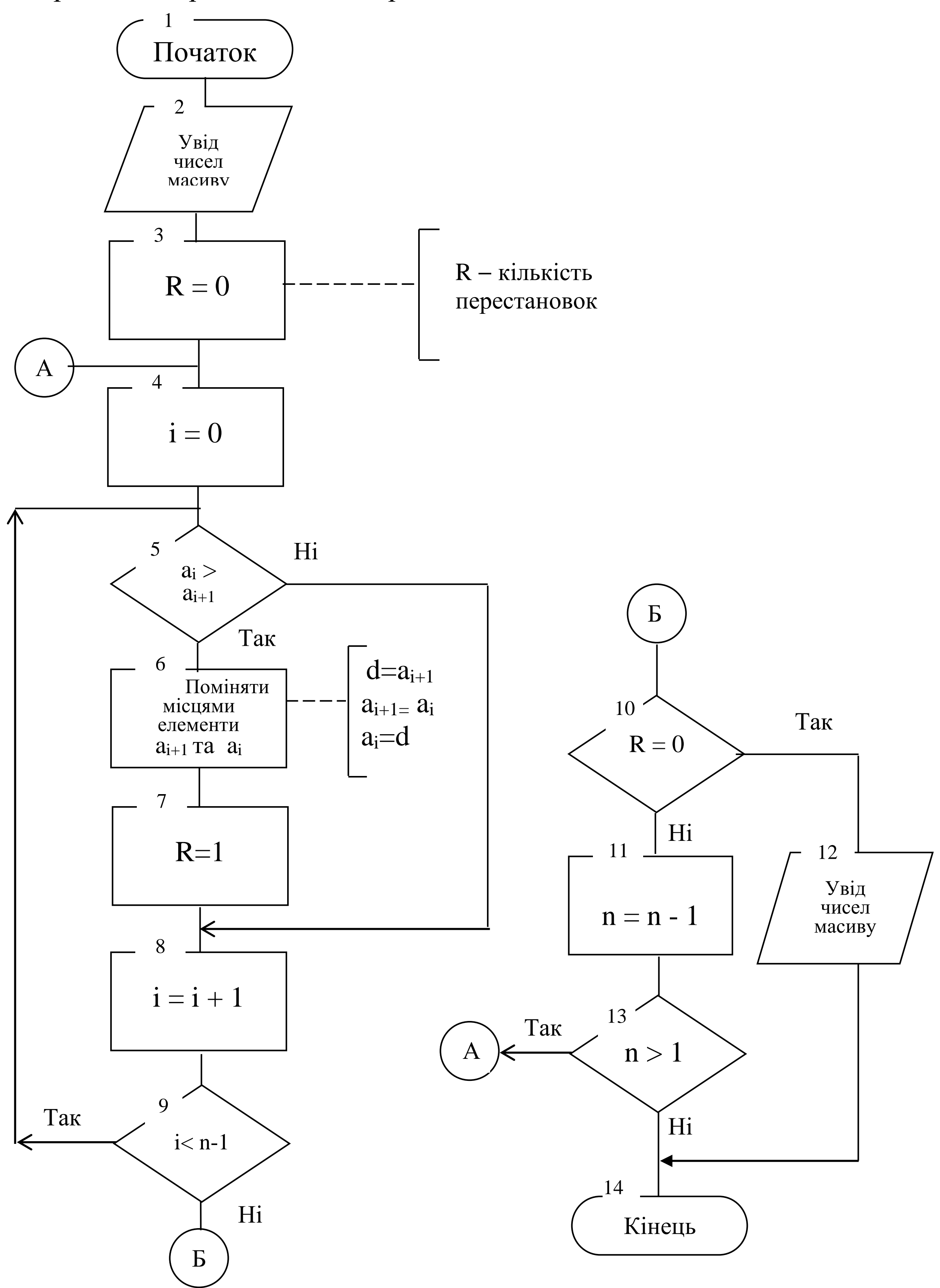



Рисунок 6.2 – Блок-схема упорядкування даних методом попарної перестановки

Приклад 6.1. Упорядкувати елементи масиву А за зростанням методом Шелла. Масив А має наступний вигляд: A={12, 3, 6, 10, 5, 14, 2, 9}

n = 8; – кількість елементів масиву

R = 0; – кількість перестановок

$$d_1 = \left[\frac{8+1}{2}\right] = 4;$$

На першому етапі порівнюються величини, що знаходяться один від одного на відстані $d_1$=4:

A={12, 3, 6, 10, 5, 14, 2, 9}

В результаті першого перегляду масив А буде мати вигляд:

A={5, 3, 2, 9, 12, 14, 6, 10}

R≠0

Продовжуємо перегляд масиву А при заданому значенні d, тому що мали місце перестановки.

A={5, 3, 2, 9, 12, 14, 6, 10}

…

Результат другого перегляду:

A={5, 3, 2, 9, 12, 14, 6, 10}    R=0

Порівняння при d = 4 перестали приводити до перестановок. Слід розрахувати нове d:

$$d_2 = \left[\frac{d_1+1}{2}\right] = \left[\frac{4+1}{2}\right] = 3$$

A={5, 3, 2, 9, 12, 14, 6, 10}

В результаті третього перегляду елементи масиву будуть перебувати в наступній послідовності:

A={5, 3, 2, 6, 10, 14, 9, 12}   R≠0

Т.к. на цьому перегляді мали місце перестановки, продовжуємо процес не змінюючи d.

A={5, 3, 2, 6, 10, 14, 9, 12}

У результаті цього перегляду всі елементи масиву A залишилися на своїх місцях, тому варто розрахувати нове d.

$$d_3 = \left[\frac{d_2 + 1}{2}\right] = \left[\frac{3+1}{2}\right] = 2$$

A={5, 3, 2, 6, 10, 14, 9, 12}

Результат перегляду масиву:

A={2, 3, 5, 6, 9, 12, 10, 14}     R≠0

Перестановки мають місце на даному перегляді, тому сортування триває при d=2:

A={2, 3, 5, 6, 9, 12, 10, 14}

Результат перегляду:

A={2, 3, 5, 6, 9, 12, 10, 14}     R=0

Тепер розрахуємо нове d:

$$d_4 = \left[\frac{d_3 + 1}{2}\right] = \left[\frac{2+1}{2}\right] = 1$$

A={2, 3, 5, 6, 9, 12, 10, 14}

У результаті перегляду маємо послідовність елементів:

A={2, 3, 5, 6, 9, 10, 12, 14}     R≠0

Мали місце перестановки, тому здійснимо перегляд масиву ще раз при d=1:

A={2, 3, 5, 6, 9, 10, 12, 14}     R=0

При d=0 кількість перестановок дорівнює нулю, тому процес сортування даного масиву завершений.

### *6.4 Обробка символьних масивів. Функції над рядками*

Символьний рядок є масивом типу char, що закінчується нуль-символом ('\0'). Наприклад:

char fio[15];

де *fio* є символьним рядком з 15 символів.

Оголосити строкову змінну можна також як покажчик:

char * s;

В цьому випадку програміст сам повинен піклуватися про виділення пам'яті під символи, що складають рядок.

Наприклад, можна зробити це так:

s = new char [чісло_сімволов_строкі + 1];

Тут +1 потрібне для зберігання нульового символу, що завершає рядок. При використанні динамічних масивів потрібно дбати про своєчасне їх звільнення операцією delete.

Однак, таку зміну можна форматувати строковою константою:

char * s = "текст_рядка";

Для введення та виведення символьного рядка призначені бібліотечні функції **gets**() та **puts**(). Для їх використання у програмі треба до програми підключити бібліотечний файл **stdio.h**.

Функція **gets()** читає символи з клавіатури доти, поки не зустріне символ нового рядка (‘\n’), що створюється при натисканні клавіші [Enter]. Функція бере всі символи до (не включаючи) символу нового рядка, приєднує до них нуль-символ (' \0 ') і розміщає рядок у змінної, котра зазначена в дужках функції. Функція **gets()** читає рядки, що містять пробіли.

Функція **puts()** виводить символьні рядки. Аргументом цієї функції є покажчик на рядок, тобто адреса першого символу в рядку. Як **puts()** дізнається, коли їй зупинитися? Вона припиняє роботу, якщо зустрічає нуль-символ. Якщо в символьному рядку нуль-символ відсутній, то **puts()** просто перебиратиме чарунки пам'яті, наступні за символьним рядком доти, поки не знайде де-небудь нуль-символ. Після того, як буде знайдений нуль-символ **puts()** замінить його символом «нового рядка». Таким чином, після виводу символьного рядка курсор переходить на новий рядок екрана.

Приклад 6.2. Використання функцій **gets()** і **puts().**

```
#include <stdio.h>
main()  {
    char name[20];          // name – символьний рядок
    puts (" Введіть ім'я ");
    gets (name);
    puts (" Гарне ім'я ");
    puts (name);
}
```

*Примітка*. Текст у лапках і імя рядка *name* є покажчиками.

*Приклад 6.3.* Увести символьний рядок. Підрахувати кількість пробілів і табуляцій в введеному рядку. Змінити символ табуляції на пробіл.

```
#include <stdio.h>
#include <windows.h>
#define MAX 81
void main ()  {
   char line[MAX];
   int i,space,tab;
   SetConsoleCP (1251);
   SetConsoleOutputCP (1251);
   puts("Введіть рядок");
   gets (line);
   for (i=space=tab=0; i<MAX; i++)  {
           if (line [i] == ' ')
                   space++;
           if (line [i] == '\t')   {
                  tab++;
                  line [i] = ' ';
           }
   }
   puts ("Кількість пробілів:");
   printf ("%d\n",space);
   puts ("Кількість табуляцій:");
   printf ("%d\n",tab);
   puts ("Змінений рядок:");
   puts(line);
}
```

У прикладі 6.3 підраховуються символи пробілів (' ') та табуляцій ('\t') у масиві символів, пойменованому *line*, і виконується заміна кожного символу табуляції на пробіл.

Спочатку *i, space* та *tab* ініціалізуються нулем. Потім **і** порівнюється з константою *MAX*. Якщо *i* менше *MAX*, то виконується тіло оператора. У залежності від значення *line[i]*, виконуються оператори ***if***. Потім до змінної **і**

додається одиниця і знову порівнюється з константою *MAX*. Тіло оператора виконується доти, поки значення **і** не стане більше *MAX*.

Більшість бібліотек мови С++ забезпечена функціями, що працюють з рядками. Розглянемо чотири найкорисніших і поширеніших: ***strlen(), strcat(), strcmp(), strcpy()***. Для використовування цих функцій необхідно підключити до програми бібліотеку ***string.h***.

Функція ***strlen()*** повертає довжину рядка, вказаного як аргумент цієї функції. Наприклад, після виконання нижченаведеної програми, змінна *n* буде мати значення довжини ім'я, яке введе користувач:

```
#include <stdio.h>
#include <string.h>
void main()  {
  char name[20];
  int n;
  puts("Введіть ім'я");
  gets(name);
  n=strlen(name);
  printf("Ім'я містить %d літер", n);
}
```

Функція ***strcat()*** об'єднує два рядки, які передаються їй як аргументи за допомогою покажчиків. Копія другого рядка приєднується до кінця першої, і це об'єднання стає новим першим рядком. Другий рядок не змінюється.

Увага! Функція не перевіряє, чи уміщається другий рядок в першому масиві. Якщо ви помилилися при виділенні пам'яті для першого масиву (рядка), то у вас виникнуть проблеми.

Наприклад, наступний фрагмент об'єднає два рядки *name* та *fam* і результат помістить в рядок *name*.

```
char name[20], fam[10];
strcat(name,fam);
```

Функція ***strcmp()*** порівнює вміст двох рядків, які передаються їй як аргументи за допомогою покажчиків, і повертає значення 0, якщо ці два рядки однакові, або різницю в кодах символів, якщо знаходить першу пару неспівпадаючих символів.

Функція ***strcpy()*** копіює рядок, на який указує другий аргумент функції, в рядок, на який указує перший аргумент функції. Нижче приведений приклад використовування функції ***strcpy()***.

```
char orig[]="Т.Г. Шевченко";
char copy[20];
strcpy(copy, orig);
```

Після виконання цього фрагмента рядок *copy* міститиме значення рядка *orig*, тобто «Т.Г. Шевченко».

Функції перетворення рядка S у число:

- ціле: int atoi (char * S);
- довге ціле: long atol (char * S);
- дійсне: double atof (char * S).

Якщо при перетворення виникне помилка дані функції повертають значення 0.

Функції перетворення числа V в рядок S:

- ціле: itoa (int V, char * S, int kod);
- довге ціле: ltoa (long V, char * S, int kod); 2 ≤kod ≤36, для відємних чисел kod = 10.

Запитання для самоконтролю

1. Дайте визначення процесу сортування.
2. Перелічіть методи упорядкування інформації.
3. У чому полягає сутність упорядкування інформації за методом вибору?
4. Чим схожі методи сортування Шелла та обмінний?
5. Чим відрізняються методи сортування Шелла та обмінний?
6. Як розрахувати за методом Шелла першу відстань між величинами, які порівнюються?
7. За методом Шелла кожна подальша відстань між величинами зменшується чи збільшується?
8. Як оголосити символьний рядок?
9. Що виконують функції gets() і puts()?
10. Чим відрізняється функція gets() від cin?
11. Як визначити довжину символьного рядка?
12. Що виконують функції strcat(), strcmp(), strcpy()?

# 7 СТРУКТУРИ

Мова С++ дозволяє створювати складні форми даних за допомогою структур. Структура – це складений об'єкт мови С++, що представляє собою сукупність логічно пов'язаних даних різного типу, об'єднаних в групу під одним ідентифікатором. Дані, що входять в цю групу називають полями.

Термін «структура» в мові С++ відповідає двом різним за змістом поняттями:

– структура – це позначення ділянки оперативної пам'яті, де розташовуються конкретні значення даних. Надалі – це структурна змінна, поля якої розташовуються в суміжних областях ОП.

– структура – це правила формування структурної змінної, якими керується компілятор при виділенні їй місця в ОП і організації доступу до її полів.

Визначення об'єктів типу структури проводиться за два кроки:

– декларація структурного типу даних, що не призводить до виділення ділянки пам'яті;

– визначення структурних змінних з виділенням для них пам'яті.

Засобом створення складних форм даних є структурний шаблон, що описує як складається структура. Наприклад, необхідно виконати опис книг, який буде включати: назву книги, її автора і ціну. Але який тип даних може містити рядки і числа одночасно і як зберігати цю інформацію роздільно? Відповіддю є – структура.

Шаблон для збереження інформації про книгу буде виглядати таким чином:

```
struct book
{
   char title [35];
   char author[20];
   float price;
};
```

Цей шаблон описує структуру, складену з двох символьних масивів (*title* та *author* ) і одної змінної дійсного типу (*price*).

Ключове слово ***struct*** визначає, що все що йде за ним, є структурою. Далі йде необов'язковий «тег» (ім'я типу структури) – слово *book*, що є скороченою міткою, що використовується для посилання на цю структуру. Наприклад:

***struct*** book libry;

Цей рядок оголошує *libry* структурою типу *book*.

У структурі є список елементів, укладений у парні фігурні дужки. Кожен елемент визначається своїм власним описом. Елементи можуть бути даними будь-якого типу, включаючи інші структури.

Як було відзначено вище, ім'я типу структури необов'язково, але його варто використовувати, якщо ви створюєте структуру, визначивши шаблон з одному місці, а фактичні змінні в іншому (як у розглянутому вище прикладі).

Процес визначення структурного шаблона і процес визначення структурної змінної можна об'єднати в один етап. У цьому випадку можна не використовувати ім'я «тега» структури. Наприклад:

```
struct                    /*без імені типу структури */
{
   char title [35];
   char author[20];
   float price;
}libry;
```

Як же звертатися до окремих елементів структури? Для цього використовується символ «.», що позначає операцію одержання елемента структури. Наприклад: *libry. price* є елементом *price* структури *libry*.

Отже, кожну книгу можна описати структурною змінною типу *book*. Для опису декількох книг (наприклад, 100 книг) буде потрібно масив таких структур. Наприклад:

struct book libry[100];

*Приклад 7.1.* Розробити алгоритм та програму для обліку інформації про книжки та пошуку книжки, ціна якої найбільша.

Згідно з принципами структурного програмування (див. п. 1.6) спочатку розглянемо поставлену перед нами задачу у цілому і складемо укрупнену блок-схему рішення (рис. 7.1), тобто виконаємо декомпозицію задачі – розбивку задачі на менші, тобто підзадачі, для яких далі будуть розроблені блок-схеми окремо в порядку ієрархічного зв'язку. Тобто задачу обліку інформації про книжки розіб’ємо на дві підзадачі:

– введення даних про книжки;

– пошук книжки з найбільшою ціною.

При написанні коду кожну підзадачу оформимо у вигляді підпрограм (функцій).

Рисунок 7.1 – Укрупнена блок-схема алгоритму рішення прикладу 7.1

Код, що реалізує укрупнений алгоритм, наведений нижче. Для прикладу тут використана константа N, яка дорівнює 10 і визначає кількість книжок.

```
#define N 10
struct book                //структура book для даних про книжку
{
   char title [35];        // title – поле структури для назви книжки
   char author[20];        // author – поле структури для автора
   int price;              // price – поле структури для ціни
}libry[N];                 // libry – масив з N структур book
   void  vvod();           // функція вводу даних про книжки
   void poisk();           // функція пошуку книжки з найбільшою ціною
void main()
{
      vvod ();
      poisk();
}
```

Алгоритми вводу даних про книжки та пошуку книжки з найбільшою ціною наведені на рисунках 7.2 і 7.3 відповідно.

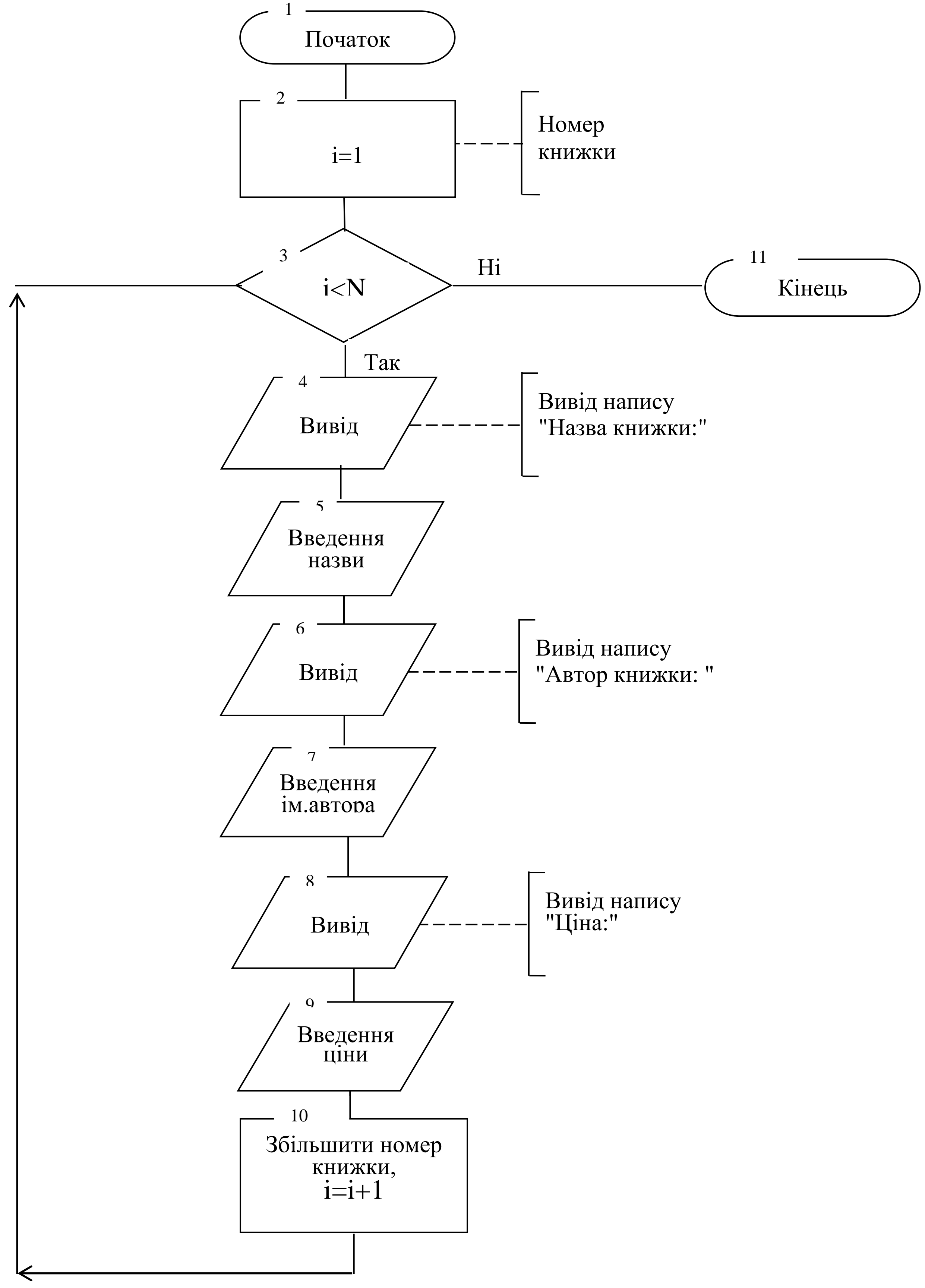



Рисунок 7.2 – Блок-схема вводу даних про книжки

Рисунок 7.2 – Блок-схема пошуку книжки з найбільшою ціною

Нижче наведений код для вирішення завдання прикладу 7.1.

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include "iostream"
using namespace std;
#define N 10
int i;
struct book
{
   char title [35];
   char author[20];
   int price;
}libry[10];

 void  vvod();              /* функція вводу даних про книжки */
 void poisk();              /* функція пошуку книжки з найбільшою ціною */

void main()
{
        SetConsoleCP (1251);
        SetConsoleOutputCP (1251);
        vvod ();
        poisk();
}

 void vvod()
 {
       for (i=0; i<N; i++)
       {
          puts ("Назва книжки: ");
          gets (libry[i].title);
          puts ("Автор книжки: ");
          gets (libry[i].author);
          puts ("Ціна:");
```

```
        cin >>libry[i].price;
      }
}

void poisk()
{
    int indmax=0;          /* індекс найбільшої ціни */
    int maxprice;          /* найбільша ціна */
    maxprice=libry[0].price;
    for (i=1; i<N; i++)
      if (libry[i].price>maxprice)
            {
              maxprice= libry[i].price;
              indmax=i;
            }
    cout << "Найбільша ціна: " <<libry[indmax].price <<"\n";
    cout << "Назва книги: " <<libry[indmax].title <<"\n";
}
```

Запитання для самоконтролю

1. Як описують структури?
2. Як описують екземпляр структури?
3. Як звертатися до окремих елементів структур?
4. Як створюються масиви структур?

# 8 ФАЙЛИ

У файлах розміщаються дані, призначені для тривалого збереження. Перш ніж одержати можливість читання чи запису інформації у файл, він повинний бути відкритий. Це можна зробити за допомогою бібліотечної функції **fopen**(). Вона бере фізичне ім'я і ставить йому у відповідності внутрішнє логічне ім'я, що використовується далі в програмах. Логічне ім'я – це покажчик на необхідний файл (на область пам'яті, де міститься інформація про файл). Покажчик на файл необхідно оголошувати у такий спосіб:

FILE * st;

де FILE – ім'я типу, що описане в стандартному визначенні stdio.h;

st – покажчик на файл (логічне ім'я). Покажчик на файл видає функція **fopen**().

st=fopen (фізичне ім'я файлу, вид використання файлу).

Вид використання файлу наведено у таблиці 8.1.

Таблиця 8.1 – Вид використання файлу

| Вид | Дія |
|---|---|
| **“r”** | Відкрити для читання |
| **“w”** | Створити для запису |
| **“a”** | Відкрити для додавання в існуючий файл |
| **“rb”** | Відкрити двійковий файл для читання |
| **“wb”** | Відкрити двійковий файл для запису |
| **“ab”** | Відкрити двійковий файл для додавання |
| **“r+”** | Відкрити файл для читання і запису |
| **“w+”** | Створити файл для читання і запису |
| **“a+”** | Відкрити для додавання чи створити для читання і запису |
| **“rt”** | Відкрити текстовий файл для читання |
| **“wt”** | Створити текстовий файл для запису |
| **“at”** | Відкрити текстовий файл для додавання |

Наприклад, якщо покажчик на файл оголошено в такий спосіб:

```
FILE * fp;
```

то для відкриття файлу з фізичним ім'ям test.dat для запису досить записати:

```
fp=fopen("test.dat", "w");
```

Але можливо, що файл на може бути відкритим (місця на диску немає чи ще що), тому можливо перевіряти відкрився файл чи ні, у такий спосіб:

```
if((fp=fopen("test.dat", "w")) ==NULL)
{
        puts("Не можу відкрити файл \n");
        exit(1);
}
```

Для читання з файлу та запису у файл блоків даних використаються функції fread()  і fwrite() відповідно.

Прототипи функцій:

```
unsigned fread(void *buf, int bytes, int c, FILE *fptr);
unsigned fwrite(void *buf, int bytes, int c, FILE *fptr);
```

де **buf** – покажчик на область пам'яті, відкіля буде відбуватися обмін інформацією;

**c** – скільки одиниць запису, кожна довжиною **bytes** байтів буде зчитано (записане);

**bytes** – довжина кожної одиниці запису в байтах;

**fptr** – покажчик на відповідний файл.

Наприклад, необхідно записати у файл відомості, що зберігаються в структурі book, яка оголошена нижче:

```
struct
{
   char title [35];
   char author[20];
   float price;
} book;

fwrite (&book, sizeof(book),1,fp);
```

де &book     – адреса структури;

sizeof(book) – розмір структури book в байтах. sizeof() – функція, яка повертає розмір, переданого їй аргументу, в байтах;

fp – покажчик на файл.

Читання і запис у файл необов'язково робити послідовно, можна це зробити безпосередньо доступом до потрібного елемента файлу за допомогою функції **fseek**(), що встановлює покажчик позиції файлу в потрібне місце. Прототип цієї функції:

int **fseek**(FILE *fptr,long numbytes, int origin);

де **fptr** – покажчик на відповідний файл;

**numbytes** – кількість байт від точки відліку для установки поточної позиції покажчика файлу,

**origin** – точка відліку – один з макросів, визначених у **stdio.h** (таблиця 8.2).

Таблиця 8.2 – Значення точки відліку

| Точка відліку | Макрос | Значення |
|---|---|---|
| Початок файлу | SEEK_SET | 0 |
| Поточна позиція | SEEK_CUR | 1 |
| Кінець файлу | SEEK_END | 2 |

Після закінчення роботи з файлом його необхідно закривати, для цього слід використовувати функцію **fclose**(), аргументом якої є покажчик на файл. Наприклад: **fclose**(fp).

Файл завжди має ознаку кінця файлу **EOF**. Кінець файлу можна визначити за допомогою функції **feof()**, аргументом якої є покажчик на файл, і повертає значення «істина», якщо досягнутий кінець файлу, і значення «неправда» – якщо не досягнутий кінець файлу. Наприклад, вираз !feof(fp) має значення істина, якщо не досягнутий кінець файлу.

*Приклад 8.1.* Уводити інформацію про книги (див. приклад 7.1) доти, поки не буде введена ознака кінця введення (наприклад, слово «Вийти»). Введену інформацію записувати у файл з ім'ям "books.dat". Вивести на екран назву і ціну книги, автора якої користувач уводитиме з клавіатури.

Програма FileBook на мові С має вигляд, наведений нижче:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
```

```
#include <stdlib.h>
#define EXIT "Вийти"
struct book
{
   char title [35];
   char author[20];
   int price;
}libry;

FILE *fbooks;
  void  vvod();
  void poisk();
void main()
{
  clrscr();
  vvod ();
  poisk();
}
void vvod()  {
       printf ("Для закінчення введення даних про книжки, на запит \"Назва
                       книги:\" введіть – Вийти \n");
     if((fbooks=fopen("books.dat", "w")) ==NULL)  {
        puts("Не можу відкрити файл \n");
        exit(1);                  /* вихід з програми*/
     }
     for (;;)             /* нескінчений цикл */
     {
          puts ("Назва книги: ");
          gets (libry[i].title);
            if (strcmp(libry.title,EXIT)==0)              // Введено „Вийти”?
            {
               fclose(fbooks);                              // Якщо введено „Вийти”
               break;                            /* вихід з нескінченого циклу */
       }
        puts ("Автор книги: ");
        gets (libry[i].author);
        puts ("Ціна:");
        cin >>libry[i].price;
```

```
        fwrite (&libry, sizeof(libry),1,fbooks); // запис у файл даних про книгу
    }
}
void poisk()  {
    char name[35];
    puts ("Введіть автора книги ");
    cin >>name;
    if((fbooks=fopen("books.dat", "r")) ==NULL)  {
        puts("Не можу відкрити файл \n");
        exit(1);
    }
    fread (&libry, sizeof(libry),1,fbooks);    /* читати з файлу*/
    while  (!feof(fbooks))                     /* перевірка на кінець файлу */
    {
        if (strcmp(libry.author, name) == 0)   /* порівняння автора,  введеного
                              на запит, з автором, який прочитали з файлу */
        {
            cout << "\n Назва книги: "<<libry.title;
            cout << "\n Ціна: "<<libry.price;
        }
        fread (&libry, sizeof(libry),1,fbooks);
    }
}
```

Запитання для самоконтролю

1. Як оголосити покажчик на файл?
2. Як відкрити файл ?
3. Яка різниця між  режимами  “r” і  “rb”  ?
4. Як записати дані у файл?
5. Як перевірити кінець файлу?

Напишіть програму, яка повинна вводити з клавіатури наступні відомості про області: назва області, її площа, кількість жителів і кількість ВУЗів, а також записувати ці відомості у файл, прочитувати їх з файлу і виводити на екран, виводити на екран відомості про області в порядку убування площі області, виводити на екран відомості по одній, введеній користувачем області. Програма повинна складатися з функцій виводу меню, введення даних, перегляду даних файлу на екрані, сортування і виводу інформації по вказаній користувачем області.

# 9 ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

## *9.1 Основні поняття об'єктно-орієнтованого програмування (ООП)*

Методологія ООП є еволюційною, тобто створеною з урахуванням технології процедурного програмування. Досвід використання процедурного (функціонального) підходу показав його основні недоліки: складність, нечіткість та непридатність для багаторазового використання. ООП усуває ці недоліки. Найскладніше щодо ООП – це перехід від мислення у термінах даних і процедур (тобто від алгоритмічного мислення) до мислення у термінах об'єктів, наділених властивостями і поведінкою.

Об'єктно-орієнтоване програмування дає новий спосіб розробки, супроводу, модернізації та повторного використання програмного забезпечення. В основі ООП лежать поняття об'єктів та класів.

У процедурному програмуванні процес перегляду складних систем спрощується за допомогою розподілу на програмні модулі, які використовуються як готові будівельні блоки. Аналогічно в ООП як блоки використовуються блоки класів та об'єктів. Основними засобами реалізації об'єктно-орієнтованого підходу є: інкапсуляція, спадкування і поліморфізм.

Ми живемо в світі об'єктів. Стіл, автомобіль, ручка, студент – все це об'єкти. Разом з фізичними існують так само абстрактні об'єкти, типовими представниками яких є числа. Таким чином, об'єкт ≡ це будь-яка фізична або абстрактна суть, що чітко ідентифікується.

Об'єкти характеризуються атрибутами. Так, атрибутами автомобіля є максимальна швидкість, потужність двигуна, колір кузова і т.д. Атрибутами студента є прізвище, ім'я, група, в якій він навчається, курс і т.д. Крім атрибутів, об'єкти володіють деякими функціональними можливостями, які в об'єктно-орієнтованому програмуванні називають операціями або методами. Так, автомобіль може їздити, корабель – плавати, комп'ютер – обчислювати.

Таким чином, об'єкт суміщає атрибути і методи, приховуючи від інших об'єктів, що взаємодіють з ним і використовують його функціональність, свою реалізацію. Так, для того, щоб перемкнути телепередачу, нам достатньо на пульті дистанційного керування набрати її номер, що запустить складний

механізм, який у результаті і призведе до бажаного результату. Нам абсолютно необов'язково знати, що відбувається в пульті дистанційного керування і телевізорі, нам лише досить знати, що телевізор володіє такою можливістю (методом), і як її можна активувати. Інкапсуляція або захования реалізації є базовою властивістю ООП. Вона дозволяє створювати призначені для користувача об'єкти, що володіють необхідними методами і далі оперувати ними, не вдаючись в будову цих об'єктів.

Об'єктно-орієнтоване програмування моделює об'єкти реального світу за допомогою програмних аналогів. ООП дає найбільш природний і інтуїтивний спосіб розгляду процесу програмування як моделювання реально існуючих об'єктів, їх атрибутів і поведінки. ООП моделює також зв'язки між об'єктами. Подібно тому, як люди посилають повідомлення, об'єкти теж зв'язуються один з одним за допомогою повідомлень.

Об'єктно-орієнтоване програмування – це методологія програмування, заснована на представленні програми у вигляді сукупності об'єктів, кожний з яких є об'єктом певного класу, а класи утворюють ієрархію спадкоємства.

Отже, між термінами клас і об'єкт існує чітка межа: клас – це опис, об'єкт – те, що створено відповідно до цього опису.

На UML – уніфікованій мові моделювання – клас відображається у вигляді прямокутника, розділеного на три частини. У першій міститься ім'я класу, в другій – атрибути, в третій – методи (рисунок 9.1).

| Клас А |
| --- |
| Атрибут 1<br>Атрибут 2<br>Атрибут 3 |
| Метод 1<br>Метод 2 |

Рисунок 9.1 – Відображення класу на UML

Основними властивостями об'єктно-орієнтованого програмування є інкапсуляція, спадкування і поліморфізм.

Інкапсуляція – це визначення класів – користувальницьких типів даних, які об'єднують свій вміст в єдиний тип і реалізовують деякі операції або методи над ним, та приховують деталі реалізації від користувача.

Спадкування – є спосіб визначення нового класу (типу) на основі того, що вже існує. При цьому успадковуються елементи (властивості і методи) того класу, що існує, і додаються нові, модифікуючи або розширюючи його.

Поліморфізм – це можливість для об'єктів різних класів, зв'язаних за допомогою спадкування, реагувати різним чином при зверненні до однієї і тієї ж функції-елемента.

Запитання для самоконтролю

1. Дайте визначення об’єкта та чим він характеризується.
2. Дайте визначення класу і як відображається він на UML.
3. Назвіть фундаментальні властивості об'єктно-орієнтованого програмування.
4. Дайте визначення інкапсуляції, спадкуванню та поліморфізму.
5. Назвіть переваги об'єктно-орієнтованого програмування.

## *9.2 Класи та структури С++ як засоби реалізації ОПП*

У більшості об'єктно-орієнтованих мов об'єкти розглядаються з погляду їхньої приналежності до класів. З цією метою після визначення об'єктів виконується їхня класифікація, тобто встановлюється, до якого класу ці об'єкти відносяться. Наприклад, чайники, холодильники, обігрівачі – це види електротоварів та можуть бути об'єднані у *клас електротоварів*.

Клас – це структурований тип даних, визначений користувачем. Клас включає, з одного боку, структуру даних, з другого – допустимі над нею операції.

Структури було розглянуто у розділі 7. Нагадаємо, наприклад, структурний шаблон для зберігання інформації про книгу буде виглядати так:

```
struct Book
{    char title [35];
     char author[20];
     float price;
};
```

Цей шаблон описує структуру, що складається з двох символьних масивів (*title* і *author*) та однієї змінної з плаваючою точкою (price).

Структура – це об'єднання одного чи більше об'єктів, які можуть бути різних типів. Як вже відомо, структура оголошується за допомогою ключового слова *struct*. Далі йде необов'язковий «тег» (ім'я типу структури) – слово *Book*, яке є скороченою міткою, яка використовується для посилання на цю структуру. Наприклад:

```
struct Book lib;
```

Цей рядок оголошує *lib* як структуру типу *Book*. Оголошення *lib* як структури типу *Book* можна записати і в такому вигляді:

```
struct Book
{
    char title [35];
    char author[20];
    float price;
}lib;
```

Слід пам'ятати, що структура містить лише дані (атрибути). Якщо дані (атрибути) містяться окремо від набору допустимих над ними функцій, зв'язку між даними та функціями немає. Але якщо ми помістимо дані та функції в одному місці, наприклад, в оголошенні структури, то отримаємо новий тип даних – *клас*. Приклад опису класу мовою C++ наведений нижче:

```
class Book
{
   char title [35];
   char author[20];
   float price;
   void input();      //функція введення даних про книгу
   void display();    //функція перегляду інформації про книгу
};
```

Отже, визначення або оголошення класу починається з ключового слова class. Далі у фігурних дужках ({}) описується тіло класу. Визначення класу закінчується крапкою з комою. Тіло класу включає дані (члени-дані), функції (члени-функції) і специфікатори доступу до членів класу. Специфікатори доступу до членів класу завжди закінчуються двокрапкою (**:**) і можуть

з'являтися у визначенні класу багато разів і у будь-якому порядку. У загальному вигляді визначення (оголошення) класу записується таким чином:

```
class <ім’я класу>
{
    [специфікатор доступу:]
     <тип атрибуту 1> < ім’я атрибуту 1 >;
     <тип атрибуту 2> < ім’я атрибуту 2>;
     ...
     [специфікатор доступу:]
     <тип функції 1> < ім’я функції 1>;
     <тип функції 2> < ім’я функції 2>;
     ....
};
```

Оголошення класу завжди закінчується крапкою з комою (**;**).

Після того, як клас визначений, і його функції-члени оголошені, ці функції-члени повинні бути визначені (описані). Кожна функція-член може бути описана прямо в тілі класу або після тіла класу.

Коли функція-член описується після відповідного визначення класу, імені функції передує ім'я класу і оператор дозволу області дії (::). Оскільки різні класи можуть мати члени з однаковими іменами, оператор дозволу області дії «прив'язує» ім'я члена до імені класу, щоб однозначно ідентифікувати функції-члени даного класу.

Хорошим стилем програмування вважається винесення оголошення класу, або інтерфейсу, у файл заголовка, який зазвичай має розширення «.h». Виконання класу, тобто визначення функцій-членів, записується у файл з розширенням «.cpp», який називається реалізацією.

Зважаючи на те що, клас – це тип даних, то після того, як клас визначений, ім'я класу може бути використано для оголошення змінних цього типу, які називають ***екземплярами класу*** або ***об'єктами***. Оголошення об’єкта (екземпляра класу) можна виконати двома способами.

*1. Створення об’єкта одночасно з оголошенням класу:*

```
class Book
{
   char title[35];
```

```
   char author[20];
   float price;
   void input();            // функція введення даних про книгу
   void display();          //функція перегляду інформації про книгу
}lib;                       // lib – об'єкт (екземпляр класу Book)
```

*2. Створення об'єкта окремо від оголошення класу:*

```
class Book
{
   char title [35];
   char author[20];
   float price;
   void input();            // функція введення даних про книгу
   void display();          //функція перегляду інформації про книгу
};
…
Book lib;          // оголошення об'єкта lib (экземпляру класу Book)
```

На основі одного класу можна визначити кілька екземплярів цього класу, тобто кілька об'єктів.

Функції, що оголошені усередині класу, викликаються ззовні об'єкта даного класу за допомогою оператора вибору елемента «**.**» або оператора вибору елемента за допомогою покажчика «**->**». Наприклад, виклик функції *input()* об'єкта *lib* класу *Book* виконується так: *lib***.***input();*

Таким чином, слід запам'ятати, що виклик функції-члена будь-якого об'єкта класу має опис:

<ім'я об'єкта> <точка> <ім'я функції, що викликається>;

Клас визначає поведінку об'єктів, які є його екземплярами. Крім того, клас служить шаблоном, на підставі якого створюються об'єкти.

## 9.3 Обмеження доступу до атрибутів та методів класу

Принцип інкапсуляції дозволяє обмежувати доступ до атрибутів та методів (функцій) класу з боку тих методів (функцій), які не є членами цього класу. У ООП класи допускають широке використання принципу інкапсуляції,

тобто приховування даних, що забезпечується видимістю (доступом до елементів класу). Тобто, при описі класу можна використовувати специфікатори доступу до членів класу. Специфікатори доступу – це спеціальні синтаксичні конструкції, що явно задають область видимості кожного члена класу. Існують наступні специфікатори доступу: public (відкритий), private (закритий), protected (захищений).

Хороший стиль програмування рекомендує використовувати при визначенні класу кожен специфікатор доступу до елементів тільки один раз, що робить програму яснішою і простішою для читання. Синтаксис використання ключових слів, що управляють доступом, є наступним:

```
class <ім'я класу>
{
    public:
    <тип відкритого атрибута 1> <ім'я відкритого атрибута 1>;
    <тип відкритого атрибута 2> <ім'я відкритого атрибута 2>;
    ...
    <тип відкритої функції 1> <ім'я відкритої функції 1>;
    <тип відкритої функції 2> <ім'я відкритої функції 2>;
    ....
    protected:
    <тип захищеного атрибута 3> <ім'я захищеного атрибута 3>;
    ...
    <тип захищеної функції 3> <ім'я захищеної функції 3>;
    ....
    private:
    <тип закритого атрибута 4> <ім'я закритого атрибута 4>;
    ...
    <тип закритої функції 4> <ім'я закритої функції 4>;
    ....
};
```

За замовчанням доступ до членів класу – private. Доступ до відкритих членів класу мають будь-які функції в програмі, доступ до закритих членів класу мають тільки члени-функції даного класу і друзі класу. Для функцій, які не є членами класу, закриті члени просто не видно. Доступ до захищених членів класу мають лише члени цього ж класу та класів-спадкоємців.

Зазвичай дані-члени класу робляться закритими, а функції-члени – відкритими. Глибокий сенс такого підходу полягає в тому, що дійсне представлення даних усередині класу не торкається клієнтів (користувачів) класу. Клієнти класу використовують клас, не знаючи внутрішніх деталей його реалізації. Якщо реалізація класу змінюється, інтерфейс класу залишається незмінним і дійсний код клієнта класу не вимагає змін. Це значно спрощує модифікацію систем.

З того, що дані класу закриті, не витікає, що клієнти не можуть змінювати ці дані. Дані можуть бути змінені функціями-членами або друзями цього класу. Наприклад, щоб дозволити клієнтам класу прочитати закрите значення даних, клас може мати функцію «одержати» (get). Щоб дати клієнтам можливість змінювати закриті дані, клас може мати функцію «встановити» (set). Таким чином, доступ до закритих членів класу здійснюється через відкриті члени-функції даного класу.

Основне завдання відкритих членів класу полягає в тому, щоб дати клієнтам класу уявлення про можливості (послуги), які забезпечує клас. Цей набір послуг складає відкритий інтерфейс класу. Клієнтів класу не повинно торкатися, яким чином клас виконує їх завдання. Закриті члени класу і опис відкритих функцій-членів недоступні для клієнтів класу. Ці компоненти складають реалізацію класу.

Наприклад, об'єкт товар характеризується двома атрибутами: найменуванням і ціною. Необхідно ввести з клавіатури дані про товар і вивести їх на екран. Отже, для об'єкта товар можна оголосити клас, що буде містити два поля даних (найменування товару й ціну), і два методи (введення даних з клавіатури і вивід даних на екран). Зобразимо клас на мові UML наступним чином:

| Товар | CGoods |
|---|---|
| Найменування<br>Ціна | name<br>price |
| Увід<br>Вивід | input()<br>output() |

Рисунок 9.2 – Зображення класу Товар на UML

Оголосимо клас CGoods:

```
class CGoods              // CGoods – ім'я класу
{
//оголошення членів-даних класу
    char name[15];       // найменування товару
    float price;                  // ціна товару
  public:                    // специфікатор доступу (відкритий)
//оголошення членів-функцій класу (методів)
    void input();         //функція введення з клавіатури даних про товар
    void output();                //функція виводу на екран даних про товар
};
```

Далі необхідно визначити (описати) члени-функції. Визначення функцій виконаємо поза класом, тому перед ім'ям функції необхідно вказати ім'я класу.

Приклад опису функції input() поза класом наведений нижче:

```
void  CGoods::input()     //оголошення функції input() класу CGoods
{                              // початок функції
    cout <<"Введіть найменування товару ";
    cin >>name;
    cout <<"Введіть ціну товару ";
    cin >>price;
}                              // кінець функції
```

Оголошення екземпляра класу або об'єкта з ім’ям Goods наведено нижче:

```
CGoods Goods;
```

Наприклад, виклик функції output() об’єкта Goods у функції main() запишеться в такий спосіб:

```
Goods.output();
```

У C++ слід дотримуватися правил при оголошенні членів класу та доступу до них, а саме:

- відкриті (public), захищені (protected) та закриті (private) члени можуть з'являтися у будь-якій послідовності у декларації класу. Переважно перераховувати відкриті, захищені та закриті члени саме в такому порядку;
- за замовчуванням для члена класу використовується закритий доступ;
- оголошення (декларація) класу може містити декілька відкритих, захищених та закритих розділів;
- оголошення класу не повинне містити всі три розділи;

– екземпляри класу (тобто об'єкти) не мають доступу до захищених та закритих членів;

– правила доступу не належать до функцій-членів самого класу;

– усі функції-члени класу мають автоматичний доступ до всіх членів-даних цього класу.

Запитання для самоконтролю

1. Дайте визначення класу та об’єкта у програмуванні.
2. Що може містити оголошення класу?
3. Як оголосити клас на мові C++?
4. Де може бути визначена (описана) функція класу?
5. Порівняйте виклик члена-функції класу у функціях класу та поза функціями цього класу.
6. Назвіть специфікатори доступу до членів класу.
7. Чи може з'являтися у визначенні класу один і той же специфікатор доступу багато разів?
8. У якому порядку можуть з'являтися специфікатори доступу до членів класу?
9. Який рівень доступу за замовчуванням для члена-класу?
10. Які функції в програмі мають доступ до відкритих членів класу?
11. Які функції в програмі мають доступ до закритих членів класу?
12. Як здійснюється доступ до закритих членів класу функціями, які не є членами цього класу?

### *9.4 Конструктори і деструктори*

В класі під час опису атрибутів не можливо надати їм значень. Для цього створюють спеціальний метод, що має назву конструктор, ім‘я якого співпадає з іменем класу. Конструктор – це метод класу, який призначений для ініціалізації (надання початкових значень) атрибутам класу. Тобто, після створення об'єкта його атрибутам можна задати початкові значення за допомогою конструктора. Конструктор – це спеціальна функція класу з тим же ім'ям, що і клас, який потім викликається автоматично під час створення екземпляра класу (об'єкта). У зв’язку з тим, що дані-члени класу не можуть

набувати початкового значення у визначенні класу, вони або повинні набути цього значення в конструкторі класу, або їх значення можна встановити пізніше, після створення об'єкта. Конструктори не можуть повертати які-небудь значення. Конструктори можна перезавантажувати, щоб забезпечити безліч початкових значень об'єктів класу, тобто клас може мати декілька конструкторів. Хороший стиль програмування свідчить: завжди передбачайте конструктор для впевненості в тому, що об'єкт набув відповідного значення, що має сенс.

Якщо для класу не визначено ніякого конструктора, компілятор автоматично вбудовує в програму конструктор без параметрів. Такий конструктор називається конструктором за замовчанням. Такий конструктор не задає ніяких початкових значень (дані об'єкта мають випадкові значення, що залишилися в оперативній пам'яті від роботи попередніх програм, тобто «сміття» оперативної пам'яті). Якщо важливо, які значення будуть мати поля (атрибути) об'єкта після його створення, то слід явно визначити конструктор.

Наприклад, оголосимо клас круг з конструктором без параметрів:

```
class  CCircle
{
  int radius;
public:
  CCircle ();// оголошення  конструктора без параметрів
};
```

Визначення (опис) конструктора наведено нижче:

```
CKrug :: CCircle ()
{
  radius=0;  // тут можна задати будь-яке значення, наприклад 10
}
```

Радіус круга ініціалізується константним значенням, в даному випадку цілим значенням 0. Об'єкти, які будуть створені за допомогою цього конструктора, завжди матимуть однакові значення (в даному випадку 0). Зручно проводити ініціалізацію об'єкта різними значеннями. Такі значення можна передавати конструктору як аргументи. Отже, додамо в клас CKrug оголошення конструктора з параметром:

```
class  CCircle  {
    int radius;
```

```
public:
   CCircle ();                    // оголошення  конструктора без параметрів
    CCircle (int rad);            // оголошення  конструктора з параметром
};
```

Визначення (опис) конструктора поза класом:

```
CCircle:: CCircle (int rad) {
  radius=rad;
}
```

При оголошенні об'єкта класу, що містить конструктор з параметрами між ім'ям об'єкта і крапкою з комою, можна в дужках вказати список ініціалізації атрибутів (членів-даних). Ці значення передаються в конструктор класу, якими ініціалізуються дані об'єкта цього класу. Наприклад:

```
CCircle Circle (25);
```

Після створення об'єкта Circle, радіус матиме значення 25.

Також можливо виконати ініціалізацію об'єкта за допомогою значень полів вже існуючого об'єкта. Для цього не потрібно самим створювати спеціальний конструктор, оскільки такий конструктор надається компілятором для кожного створюваного класу і називається конструктором копіювання за замовчанням. Конструктор копіювання має єдиний аргумент, що є об'єктом того ж класу, що і конструктор.

Приклад використання конструктора копіювання демонструє наступний фрагмент:

```
CCircle Circle1(40);
CCircle Circle2(Circle1);
```

Радіус об'єкта Circle1, після його створення, буде мати значення 40. Радіус об'єкта Circle2, також буде мати значення 40, бо його ініціалізація відбулася значенням об'єкта Circle1 за допомогою конструктора копіювання. Тобто, дія конструктора копіювання зводиться до копіювання атрибутів одного об'єкта у відповідні атрибути другого об'єкта.

Якщо в класі оголошено декілька конструкторів, то відбувається виклик того конструктора, у якого кількість і типи аргументів відповідають кількості і типам, вказаним при створенні об'єкта (екземпляра класу).

Подібно до того, як конструктори створюють об'єкт, деструктори руйнують його тоді, коли він більше не потрібен. Тобто, деструктор – це також

спеціальна функція класу, що використовується для звільнення пам'яті, яку займає об'єкт. Ім'я деструктора співпадає з ім'ям класу, але перед ним ставиться символ тильда (~). Деструктор класу викликається при руйнуванні об'єкта – наприклад, коли виконувана програма покидає область дії, в якій був створений об'єкт цього класу. Насправді деструктор сам не руйнує об'єкт – він виконує підготовку перед тим, як система звільняє область пам'яті, в якій зберігався об'єкт, щоб використовувати її для розміщення нових об'єктів.

Якщо деструктор явним чином не визначений, компілятор автоматично створює порожній деструктор. Описувати в класі деструктор явним чином потрібно у разі, коли об'єкт містить покажчики на пам'ять, що виділяється динамічно – інакше при руйнуванні об'єкта пам'ять, на яку посилалися його поля-покажчики, не буде помічена як вільна. Деструктор не приймає ніяких параметрів і не повертає ніяких значень. Клас може мати тільки один деструктор – перевантаження деструктора не дозволяється, або не мати жодного. У такому випадку компілятор сам створює деструктор за замовчуванням;

Додамо в клас CCircle оголошення деструктора:

```
class  CCircle
{
    int radius;
public:
   CCircle ();                    // оголошення  конструктора без параметрів
   CCircle (int rad);             // оголошення  конструктора з параметром
  ~ CCircle ();                   // оголошення  деструктора
};
```

Визначення (опис) деструктора поза класом:

```
CCircle::~ CCircle ()
{
        // код для звільнення динамічної пам'яті
}
```

Розглянемо приклад програми з використанням кількох конструкторів, а також відкритих членів-функцій класу, за допомогою яких здійснюється доступ до закритих членів-даних класу.

```
class CPoint // початок оголошення класу для точки с координатами x, y
{
```

```
    double x;                // координата x
    double y;                // координата y
public:
// оголошення конструкторів
    CPoint();                          // конструктор без параметрів
    CPoint(double a, double b);        // конструктор с параметрами
    CPoint(CPoint &P);                 // конструктор копіювання
// оголошення функцій, що змінюють значення координат x і y відповідно
    void  SetX(double fx);
    void  SetY(double fy);
// оголошення і визначення функцій, що дозволяють отримати значення
// координат x і y відповідно
    double GetX() { return x; };
    double GetY() { return y; };
// оголошення функції, що розраховує відстань від точки до початку
// координат
    double DistanceCoordinate();
// оголошення функції, що розраховує відстань між двома точками
    double DistancePoint(CPoint &P);
};              // кінець оголошення класу для точки
// визначення конструктора без параметрів
CPoint::CPoint() {
        x=1;
        y=1;
}
// визначення конструктора с параметрами
CPoint::CPoint(double a, double b) {
   x=a;
   y=b;
}
// визначення конструктора копіювання
CPoint::CPoint(CPoint &P)  {
   x=P.x;
   y=P.y;
}
```

```
// визначення функції, що змінює значення координати x
void  CPoint::SetX(double fx)  {
        x=fx;
}
// визначення функції, що змінює значення координати y
void  CPoint::SetY(double fy)  {
   y=fy;
}
// визначення функції, що розраховує відстань від точки до початку
// координат
double CPoint::DistanceCoordinate()  {
    return sqrt (pow (x,2)+ pow (y,2));
  }
// визначення функції, що розраховує відстань між двома точками
double CPoint::DistancePoint(CPoint &P)  {
         return sqrt(pow (x-P.x, 2)+ pow (y-P.y, 2));
}
void main()  {
   double x2=10.5;
   double y2=45.6;
   CPoint PointA();
   CPoint PointB(PointA);
   CPoint PointC(3,5);
   PointA.SetX(x2);
   PointA.SetY(y2);
   cout<<"Точка А має координати: "<<PointA.GetX() <<","
         <<PointA.GetY()<<"\n";
   cout<<"Точка В має координати: "<<PointB.GetX() <<","
         <<PointB.GetY()<<"\n";
   cout<<"Точка С має координати: "<<PointC.GetX() <<","
      <<PointC.GetY()<<"\n";
   cout<<"Відстань від точки А до початку координат:"
      <<PointA.DistanceKoord()<<"\n";
   cout<<"Відстань від точки В до початку координат:"
      <<PointB.DistanceKoord()<<"\n";
```

```
    cout<<" Відстань між точками А і В: "
        <<PointA.DistancePoint(PointB)<<"\n";
}
```

У наведеному прикладі оголошено клас CPoint, який має три конструктори: CPoint(), CPoint(CPoint &P), CPoint(double a, double b).

Конструктор без параметрів CPoint() використовується для ініціалізації об'єкта завжди одними й тими ж значеннями, а саме координата *x* і координата *y* будуть матиме значення 1.

Конструктор копіювання CPoint(CPoint &P) використовується для ініціалізації координат створюваного значеннями об'єкта, що стоїть у дужках цього конструктора шляхом копіювання його координат.

Конструктор із параметрами CPoint(double a, double b) використовується для ініціалізації координат створюваного об'єкта значеннями типу double, що вказані у дужках цього конструктора при оголошенні об'єкта.

У функції main() оголошуються та ініціалізуються локальні змінні x2 та y2 значеннями 10,5 та 45,6, а також оголошуються наступні екземпляри класу CPoint, а саме:

− об'єкт PointA, який оголошено так: CPoint PointA;. Об'єкт PointA ініціалізується за допомогою конструктора без параметрів, оскільки оголошення об'єкта PointA не має жодних параметрів. В результаті роботи конструктора об'єкт PointA матиме координати x=1 і y=1 (дивись код конструктора без параметрів CPoint();

− об'єкт Point, який оголошено так: CPoint PointB(PointA);. Об'єкт PointB ініціалізується за допомогою конструктора копіювання, оскільки оголошення об'єкта PointB має як параметр об'єкт PointA. В результаті роботи конструктора копіювання об'єкт PointB буде мати ті ж координати, які має об'єкт PointA, а саме x = 1 і  y=1;

− об'єкт PointC, який оголошено так: CPoint PointC(3, 5);. Об'єкт PointC ініціалізується за допомогою конструктора з параметрами, оскільки оголошення об'єкта PointC має як параметри два числа. В результаті роботи конструктора з параметрами об'єкт PointC матиме ті координати, які передані конструктору як параметри, а саме: x=3; y=5.

Після оголошення об'єкта PointC у функції main() відбувається виклик функцій SetX(x2) та SetY(y2) для об'єкта PointA наступним чином:

```
PointA.SetX(x2);
```

PointA.SetY(y2);

В результаті роботи цих функцій у об'єкті PointA буде змінено значення координат *x* і *y* на значення змінних *x2* і *y2* відповідно. Тобто об'єкт PointA буде матиме координати *x*=10.5; *y*=45.6.

Доступ до закритих членів класу здійснюється через відкриті члени функції цього класу. Функції SetX(double fx) и SetY(double fy) змінюють значення координат *x* та *y* на значення змінних *fx* і *fy* відповідно того об'єкта, для якого вони викликані. Наприклад, маємо у програмі наступні рядки:

PointA.SetX(x2);

PointB.SetX(x2);

В першому випадку функція SetX() змінює координату *x* об'єкта PointA значенням *x2*, а в другому – змінює теж координату *x* але об'єкта PointB.

Функції GetX() та GetY() повертають значення координат *x* та *y* того об'єкта, для якого вони викликані. Наприклад:

PointC.GetY(); – цей рядок означає, що функція GetY() поверне значення координати *y* об'єкта PointC.

Рядок PointA.DistanceKoord() означає виклик функції DistanceKoord() об'єкта PointA, в результаті чого буде розрахована відстань від точки PointA до початку координат. Для визначення відстані від точки PointB до початку координат необхідно записати PointB.DistanceKoord().

Рядок PointA.DistancePoint(PointB) дозволяє визначити відстань між точками PointA і Point B.

*Завдання для самостійного виконання:*

Додайте у попередню програму за допомогою класу CPoint створення об'єкта PointN, координати якого дорівнюють нулю. Вивести на екран значення координат об'єкта PointN. Змінити координату x на 5, а координату y на 8. Розрахувати й вивести на екран відстань від точки PointN до початку координат, а також відстань між точками PointN і PointC.

Запитання для самоконтролю

1. Дайте визначення конструктора класу.
2. Для чого призначений конструктор класу?
3. Які види конструкторів ви знаєте?
4. Скільки конструкторів можна оголосити у класі?

5. Як у класі оголосити конструктор з параметрами?
6. Для чого призначений деструктор класу?
7. Скільки деструкторів можна оголосити у класі?
8. Як викликається конструктор і деструктор?
9. Якщо клас має декілька конструкторів, то який з конструкторів викликається при створенні об'єкта цього класу?
10. Чи завжди необхідно описувати в класі деструктор? Якщо ні, то в яких випадках?

### *9.5 Спадкування*

Об'єктно-орієнтоване програмування дає змогу оголошувати нові класи, які є нащадками існуючих класів. Можливість успадкування дозволяє класам-нащадкам робити більш спеціалізовані дії, ніж ті, які виконують їх батьківські класи. Тобто, спадкування – процес створення нових класів, званих нащадками або похідними класами, з вже існуючих, або базових, класів. Похідний клас отримує всі можливості базового класу, але може бути вдосконалений за рахунок додавання власних. Базовий клас при цьому залишається незмінним. Повторне використання кодів заощаджує час при розробці програм. Це сприяє повторному використанню перевіреного і налагодженого коду і таким чином зменшує число проблем, що виникають після того, як система починає функціонувати.

Похідний клас в загальному випадку більший за свій базовий клас. Похідний клас більш специфічний по своєму призначенню, вужчий, чим його базовий клас, і представляє меншу групу об'єктів. Кожен об'єкт похідного класу є також об'єктом відповідного базового класу. Проте, зворотне невірно: об'єкт базового класу не є об'єктом класів, породжених цим базовим класом.

Наприклад, *прямокутник* є *чотирикутником*, але невірно стверджувати, що *чотирикутник* є *прямокутником*. В цьому випадку клас *чотирикутник* є базовим класом, а клас *прямокутник* – спадкоємцем класу або похідним класом.

Від будь-якого класу можна породити один або кілька підкласів (похідних класів), Похідні класи можуть бути базовими класами для інших похідних класів, у результаті чого сформується ієрархія класів. Батьківські

класи зазвичай містять методи більш загального характеру, тоді як рішення специфічних завдань поручається похідним класам.

Наведемо приклад ієрархії спадкування. Типовий заклад вищої освіти (ЗВО) складається з тисячі людей, які є його членами. Ці люди діляться на студентів і службовців. Службовці – або члени факультету, або допоміжний персонал. Члени факультету – або адміністратори (декани і завідувачі кафедр), або викладачі. Це дає ієрархію спадкування, яка показана на рисунку 9.3.

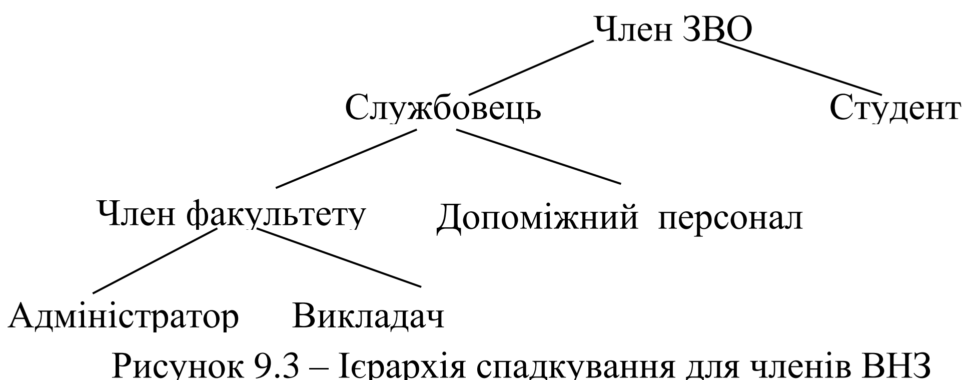
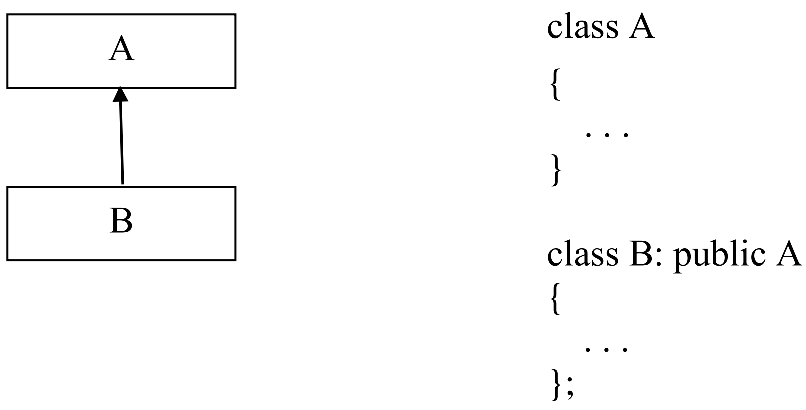



Рисунок 9.3 – Ієрархія спадкування для членів ВНЗ

У С++ підтримуються дві схеми спадкування: одиночне спадкування та множинне спадкування. При одиночному (простому) спадкуванні клас-нащадок породжується від єдиного батьківського класу (рисунок 9.4). При множинному спадкуванні клас-нащадок породжується від двох і більше батьківських класів (рисунок 9.5). Множинне спадкування дозволяє одному класу володіти властивостями двох і більш батьківських класів. Спадкування в UML називають узагальнення, і на діаграмі класів його зображають у вигляді стрілки. (Діаграма класів описує структуру системи, показуючи її класи, їх атрибути і методи, а також відносини (зв'язки), які існують між ними).




```
class A
{
   . . .
}

class B: public A
{
   . . .
};
```

Рисунок 9.4 – Одиночне спадкування на UML та С++

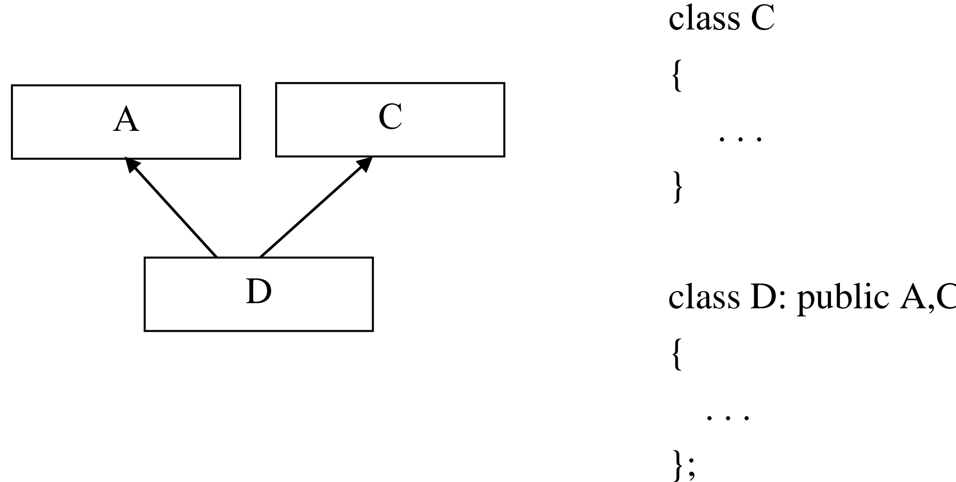



Рисунок 9.5 – Множинне спадкування на UML та C++

При описі класу в його заголовку перераховуються всі класи, що є для нього базовими. Можливість звернення до елементів цих класів регулюється за допомогою специфікаторів доступу private, protected і public. Якщо базових класів декілька, вони перераховуються через кому. Специфікатор доступу може стояти перед кожним класом. За замовчанням для класів використовується ключ доступу private. Тобто, спадкування може бути відкрите, закрите й захищене. У наведеному прикладі клас В оголошується похідним від класу А, спадкування – відкрите. При відкритому спадкуванні відкриті й захищені елементи базового класу успадковуються як відкриті й захищені елементи похідного класу відповідно.

Похідні класи успадковують члени-дані й члени-функції базового класу. Конструктори не успадковуються, тому похідний клас повинен мати власні конструктори, деструктори успадковуються. Порядок виклику конструкторів визначається наведеними нижче правилами:

– якщо в конструкторі похідного класу явний виклик конструктора базового класу відсутній, автоматично викликається конструктор базового класу за замовчанням (тобто той, який можна викликати без параметрів);

– для ієрархії, що складається з декількох рівнів, конструктори базових класів викликаються, починаючи з самого верхнього рівня. Після цього виконуються конструктори тих елементів класу, які є об'єктами, у порядку їх оголошення в класі, а потім виконується конструктор класу;

– у разі декількох базових класів їх конструктори викликаються у порядку оголошення.

*Зауваження*. Якщо конструктор базового класу вимагає вказівки параметрів, він повинен бути явним чином викликаний в конструкторі похідного класу у списку ініціалізації.

Правила спадкування деструкторів:

– якщо в похідному класі не описано деструктор, то він формується за замовчанням і викликає деструктори всіх базових класів;

– на відміну від конструкторів, при написанні деструктора похідного класу в ньому не потрібно явно викликати деструктори базових класів, оскільки це буде зроблено автоматично;

– для ієрархії класів, що складається з декількох рівнів, деструктори викликаються в порядку, строго зворотному виклику конструкторів: спочатку викликається деструктор класу, потім – деструктор елементів класу, а потім – деструктор базового класу.

Похідний клас не може мати доступ до закритих елементів свого базового класу (дозвіл такого доступу був би порушенням інкапсуляції базового класу), але має доступ до відкритих і захищених елементів свого базового класу. Похідний клас може мати доступ до закритих елементів свого базового класу тільки за допомогою функцій доступу, передбачених у відкритому інтерфейсі базового класу.

## Запитання для самоконтролю

1. У чому полягає суть спадкування?
2. У чому переваги використання спадкування?
3. Які ви знаєте види спадкування?
4. Як оголосити похідний клас при одиночному та множинному спадкуванні?
5. Які члени успадковують похідні класи?
6. Який порядок виклику конструкторів при спадкуванні?
7. Який порядок виклику деструкторів при спадкуванні?
8. Як на UML позначається спадкування?
9. До яких елементів свого базового класу може мати доступ похідний клас?

### *9.6 Поліморфізм*

Для похідного класу можна визначати методи (функції), що мають такі ж імена, як і у методів базового класу. В цьому випадку має місце перевантаження функцій. Таким чином, похідний клас може не тільки доповнювати, але і коректувати поведінку базового класу. Доступ до перевизначеного (перевантаженого) методу базового класу для похідного класу виконується через ім'я, уточнене за допомогою операції доступу до області видимості (::).

Наприклад:

```
class A
{
   . . .
public:
  void draw();
}

class B: public A
{
   . . .
  void draw();                // перевизначення методу draw() базового класу

};
void draw();
{
   class A:: draw();        //виклик функції draw() базового класу A
   . . .
}
```

Поліморфізм – можливість для об'єктів різних класів, пов'язаних за допомогою спадкування, реагувати по-різному при зверненні до однієї і тієї ж функції-елемента. Поліморфізм реалізується за допомогою віртуальних функцій, тобто при оголошенні функції у класі використовується ключове слово virtual, наприклад:

virtual void draw();

Якщо функція один раз оголошена віртуальною, то вона залишається віртуальною на кожному більш низькому рівні ієрархічної структури. Якщо в похідному класі вирішено не описувати віртуальну функцію, то похідний клас безпосередньо успадковує опис віртуальної функції з базового класу.

Робота з об'єктами частіше проводиться через покажчики. Покажчику на базовий клас можна присвоїти значення адреси об'єкта будь-якого похідного класу. Якщо при використанні віртуальної функції виклик здійснюється за допомогою покажчика базового класу (або посилання), то C++ вибирає правильну перевизначену функцію у відповідному похідному класі, пов'язаному з даним об'єктом. Якщо функція-елемент визначена у базовому класі не як віртуальна, але перевизначена в похідному класі, і викликається через покажчик базового класу, то використовується функція базового класу. Якщо ж ця функція-елемент викликається через покажчик похідного класу, то використовується функція похідного класу. Це не поліморфне поводження.

Приклад виклику функції draw() через покажчик на базовий клас:

```
B  b;              // оголошення об’єкта похідного класу з ім’ям  b
A *pA;             // оголошення  покажчика pA на базовий клас A
pA =& b;           // покажчику привласнюється адреса об'єкта b
pA-> draw ();      // виклик функції draw() через покажчик на базовий клас
```

Отже, завдяки використанню віртуальних функцій і поліморфізму, один і той же виклик функції може призвести до різних дій залежно від типу об'єкта, що приймає цей виклик.

Розглянемо використання спадкування та поліморфізму на прикладі. Розробимо програму для виводу даних про кредити на житло та про кредити на авто. Про кредити на авто маємо наступні дані: код кредиту, ПІБ клієнта, дата відкриття кредиту, сума кредиту, термін кредиту, відсоткова ставка кредиту, марка авто, колір авто. Про кредити на житло маємо наступні дані: код кредиту, ПІБ клієнта, дата відкриття кредиту, сума кредиту, термін кредиту, відсоткова ставка кредиту, адреса житла, загальна площа.

Виходячи з умови завдання маємо два об’єкти: кредит на житло та кредит на авто. Якщо для цих об’єктів ми оголосимо два класи з тими атрибутами, що перелічені у завдані, то побачимо, що атрибути код кредиту, ПІБ клієнта, дата відкриття кредиту, сума кредиту, термін кредиту, відсоткова ставка кредиту повторюються для обох об’єктів. Тому створімо не 2 класи, а три. Першому дамо ім’я *Кредит* і оголосимо в ньому ті атрибути, які є спільними для кредиту на житло і для кредиту на авто. Другому класу дамо ім’я *Кредит на авто*, третьому – *Кредит на житло* й оголосимо їх похідними від класу *Кредит.*

Зобразимо класи та їх взаємозв’язки у вигляді діаграми класів на мові UML на рисунку 9.6. На діаграмах # позначає захищений (protected), + позначає відкритий (public),  – позначає закритий (private) доступ до членів класу.

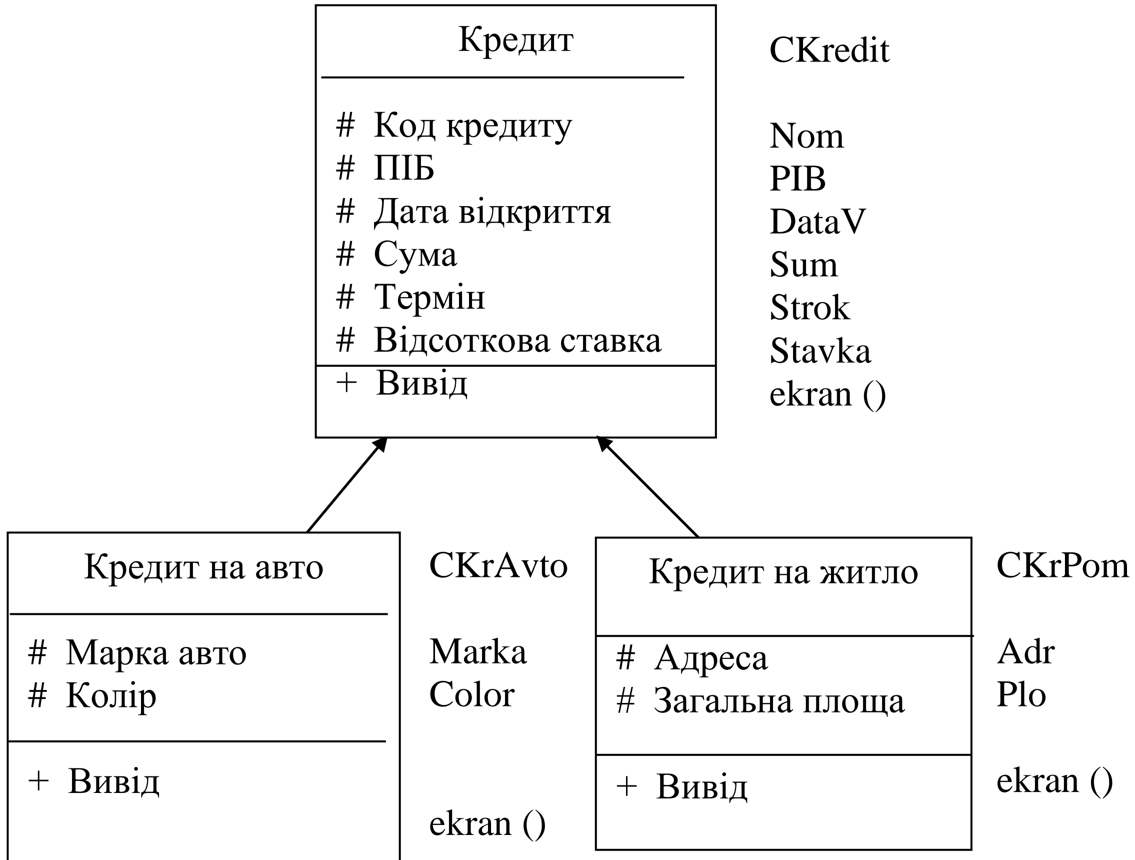



Рисунок 9.6 – Діаграма класів

У кожному класі оголосимо конструктор з параметрами, функцію виводу на екран усіх атрибутів об’єкта. Створимо об'єкти похідних класів й виконаємо ініціалізацію цих об'єктів довільними даними, використовуючи конструктори з параметрами, а також виведімо на екран атрибути всіх об’єктів. Програму за завданням наведено нижче.

```
#include <string.h>
#include <windows.h>
#include "iostream"
using namespace std;
class CKredit
{
  protected:
```

```
        int Nom;
        char PIB[40];
        char DataV[20];
        float Sum;
        int Strok;
        float Stavka;
  public:
      Kredit(int,char*,char*,float,int,float);     //конструктор з параметрами
      virtual void ekran();             //функція виводу даних на екран
};
class CKrAvto:public CKredit            //похідний клас – Кредит на авто
{
         protected:
         char Marka[15];
         char Color[20];
  public:
         CKrAvto(int,char*,char*, float,int,float,char*,char*);      // конструктор
         void ekran();
};
class CKrPom:public CKredit             //похідний клас – Кредит на житло
{
  protected:
         char Adr[35];
         float Plo;
  public:
         CKrPom(int,char*,char*,float,int,float, char*, float);       // конструктор
         void ekran();
};
      //опис конструктора
CKredit:: CKredit(int n,char* p,char* d,float s,int str,float stv)
{
         Nom=n;
         strcpy(PIB,p);
         strcpy(DataV,d);
         Sum=s;
```

```
        Strok=str;
        Stavka=stv;
}
//опис  віртуальної функції ekran()
void CKredit::ekran()
{
       cout<<"\n\n Код кредиту: "<<Nom;
       cout<<"\n ПІБ    : "<<PIB;
       cout<<"\n Дата відкриття : "<<DataV;
       cout<<"\n Сума  : "<<Sum;
       cout<<"\n Термін  : "<<Strok;
       cout<<"\n % ставка  : "<<Stavka;
};

 CKrAvto::CKrAvto(int n,char* p,char* d,float s,int str,float stv, char* m, char* c):
                                    CKredit(n,p,d, s,str,stv)
{
        strcpy(Marka,m);
        strcpy(Color,c);
}
void CKrAvto::ekran()
{
     CKredit::ekran();
     cout<<"\n Марка авто: "<<Marka;
     cout<<"\n Колір: "<<Color<<"\n";
};
CKrPom::CKrPom(int n,char* p,char* d,float s,int str,float stv, char* ad, float pl):
                                    CKredit(n,p,d, s,str,stv)
{
     strcpy(Adr,ad);
     Plo=pl;
}
void CKrPom::ekran()
{
     CKredit::ekran();
```

```
    cout<<"\n Адреса: "<<Adr;
    cout<<"\n Загальна площа: "<<Plo<<"\n\n";
};
void fn(CKredit &kr)
{
    kr.ekran();
}
int main()  {
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
CKrAvto KrAvto(11030,"Кравченко А.П.","12.12.2013",100000,36,15.5,
                    "Ford","Чорний");
CKrPom KrPom(25111,"Гурьева С.Я.","05.11.2012",30000,60,16.5,
                    "Серова,4/54",85.6);
        fn(KrAvto);
        fn(KrPom);
        system("PAUSE");
        return 0;
}
```

Результати виконання програми наведено на рисунку 9.7.

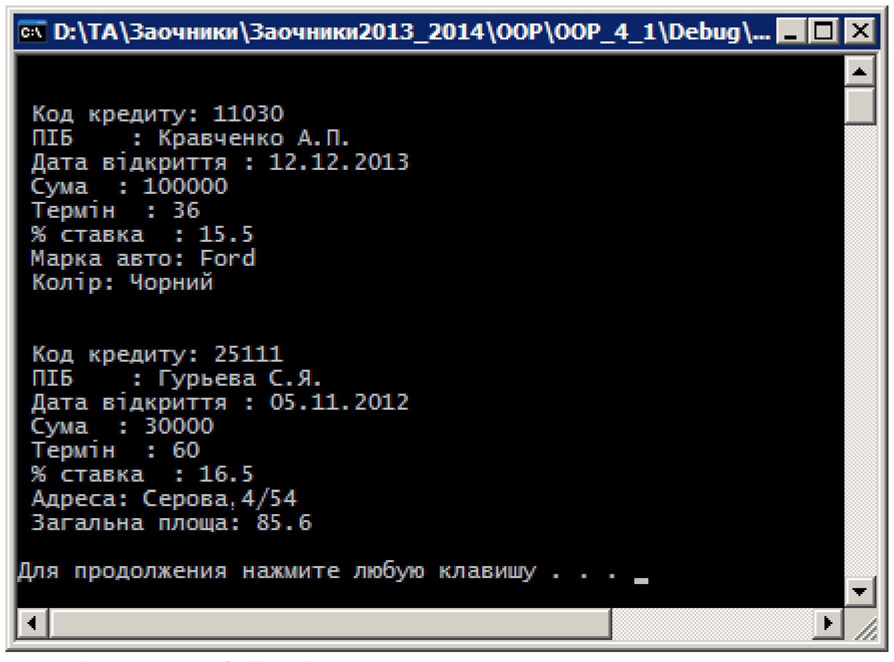



Рисунок 9.7 – Результати виконання програми

Запитання для самоконтролю

1. Дайте визначення перевантаження функцій.
2. Що означає поліморфізм?
3. За допомогою чого реалізується поліморфізм?
4. Чи можна покажчику на базовий клас присвоїти значення адреси об'єкта похідного класу?
5. Що необхідно для того, щоб реалізувати поліморфізм в програмі?

## *9.7 Абстрактні класи*

Клас є абстрактним, якщо він має хоча б одну чисту віртуальну функцію. Чистою віртуальною функцією є така функція, у якої в її оголошенні тіло визначене як 0. Наприклад:

virtual float Func()=0;

Абстрактні класи призначені для представлення загальних понять, які передбачається конкретизувати в похідних класах. Абстрактний клас може використовуватися тільки як базовий для інших класів – об'єкти абстрактного класу створювати не можна, оскільки прямий або непрямий виклик чисто віртуального методу призводить до помилки при виконанні.

Якщо клас, похідний від абстрактного класу, не визначає всі чисто віртуальні функції, він також є абстрактним.

Створювати функцію, параметром якої є покажчик на абстрактний клас, можна. На місце цього параметра при виконанні програми може передаватися покажчик на об'єкт будь-якого похідного класу. Це дозволяє створювати поліморфні функції, що працюють з об'єктом будь-якого типу в межах однієї ієрархії.

Запитання для самоконтролю

1. Дайте визначення абстрактного класу.
2. Чи можна створювати об’єкти абстрактного класу?
3. Виявіть відмінності між віртуальними функціями і чистими віртуальними функціями.
4. Чи вірно, що всі віртуальні функції в абстрактному базовому класі повинні бути оголошені як чисті віртуальні функції?

5. Чи можна створювати функцію, параметром якої є покажчик на абстрактний клас?

### *9.8 Класи вводу-виводу*

Потік – це абстрактне поняття, що відноситься до будь-якого перенесення даних від джерела до приймача. За видом пристроїв, з якими працює потік, можна розділити потоки на стандартні, файлові і строкові.

Стандартні потоки призначені для передачі даних від клавіатури і на екран дисплея. Файлові потоки для обміну інформацією з файлами на зовнішніх носіях даних (наприклад, дисках), а строкові потоки – для роботи з масивами символів в оперативній пам'яті.

Для створення стандартних потоків у мові С++ існують наступні класи:

istream – клас вхідних потоків;

ostream – клас вихідних потоків;

iostream – клас двонаправлених (вхідних і вихідних) потоків.

Клас ios є базовим класом для потоків. Частину ієрархії класів потоків вводу/виводу з класами обробки файлів наведено на рисунку 9.8.

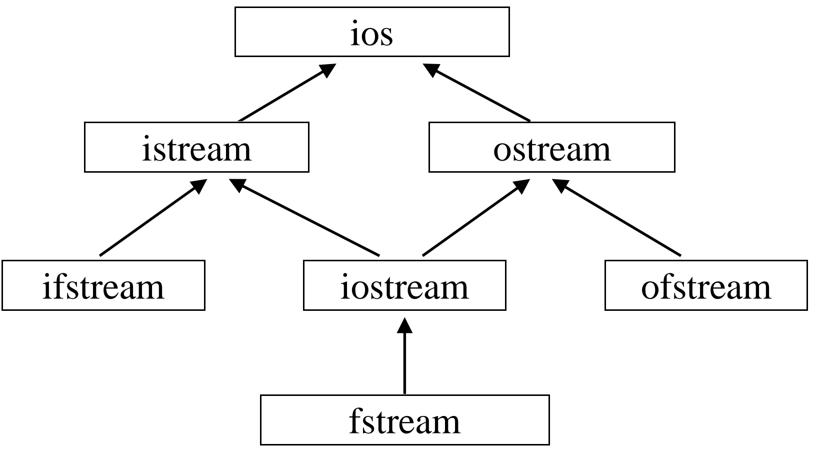



Рисунок 9.8 – Частина ієрархії класів потоків вводу/виводу з класами обробки файлів

Заголовний файл <iostream.h> містить, окрім опису класів введення/виведення, чотири об'єкти (табл. 9.1).

Таблиця 9.1 – Об'єкти, які оголошені у файлі <iostream.h>

| Об'єкт (потік) | Клас | Опис |
|---|---|---|
| cin | istream | Зв'язується з клавіатурою (стандартним буферизуючим введенням) |
| cout | ostream | Зв'язується з екраном (стандартним буферизуючим виводом) |
| cerr | ostream | Зв'язується з екраном (стандартним небуферизуючим виводом), куди направляються повідомлення про помилки |
| clog | ostream | Зв'язується з екраном (стандартним буферизуючим виводом), куди направляються повідомлення про помилки |

Для позначення виводу використовується операція (<<), яка називається операцією ***помістити в потік***. Для позначення введення використовується операція (>>), яка називається операцією ***узяти з потоку***. Ці операції застосовуються до об'єктів стандартних потоків cin, cout, cerr, clog і зазвичай використовуються також з об'єктами потоків, тип яких визначений користувачем. Для використання у програмі стандартних потоків необхідно включати файл iostream.h.

Кожна з операцій << і >> може бути використана у зчепленій формі. Багатократні операції ***помістити в потік*** або ***узяти з потоку*** виконуються в тій послідовності, в якій вони записані. Наприклад:

```
int x=10;
cout << «Змінна x має значення » << x ;    // зчеплена форма операції <<
```

Другий рядок слід читати таким чином: помістити в стандартний потік виводу (тобто вивести на екран) напис «Змінна x має значення » і помістити в стандартний потік виводу значення змінної x.

```
int c;
float b;
cin>>c>>b;
```

Третій рядок слід читати таким чином: узяти із стандартного потоку введення (тобто ввести з клавіатури) два значення і помістити перше в змінну c, а друге – в змінну b.

В ланцюжок операцій поміщення в потік і одержання з потоку можна включати функції, які називаються маніпуляторами. Маніпулятори потоків вирішують задачі форматування, тобто завдання ширини полів, завдання точності, вставку у вихідний потік символу нового рядка і т.д. Використання маніпуляторів вимагає включення заголовного файлу <iomanip.h>. Наприклад:

```
cout << «c=» << c <<endl << «b=» << b;
```

Маніпулятор endl вставляє у вихідний потік символ нового рядка, тому напис «b=» буде виведено на новому рядку.

Управляти точністю чисел з комою, що виводяться, тобто числом розрядів праворуч від десяткової крапки, можна за допомогою маніпулятора setprecision. Точність задається в дужках. Точність 0 має особливе значення, вона відновлює точність за замовчанням, що дорівнює 6. Встановлення точності діє для всіх подальших операцій виводу до тих пір, поки не буде проведено наступне встановлення точності.

```
float m=2.1456;
cout << «m=» << setprecision(2) << m;
```

У результаті виконання попереднього коду на екран виведеться: m=2.14.

Для встановлення ширини поля при виводі можна використовувати маніпулятор потоку setw. Встановлення ширини поля впливає тільки на наступну операцію ***помістити в потік*** або ***узяти з потоку***, потім ширина поля встановлюється на 0, тобто поля для представлення вихідних значень будуть такої ширини, яка необхідна для їх виводу.

```
int m=10;
cout << setw(10)  << «m=» << m;
```

Маніпулятор setw(10) встановить ширину поля для виводу напису «m=», що дорівнює 10 символам, а для змінної m – ширину 0, але виведе значення змінної на ширину 2 символів, оскільки значення змінної m складається з двох цифр (m =10).

Якщо ширина поля, що задана за допомогою setw, більше, ніж ширина значення, що виводиться, то зайві позиції за замовчанням заповнюються пропусками.

Маніпулятор setiosflags() використовується для заповнення зайвих позицій поля зліва або праворуч від значення, залежно від значення, заданого в дужках:

– значення ios::left – заповнює зайві позиції після виведеного значення, тобто відбувається вирівнювання значення по лівому краю поля;

– значення ios:: right – заповнює зайві позиції перед виведенням значення, тобто відбувається вирівнювання значення по правому краю поля.

За замовчанням встановлено вирівнювання виведеного значення по лівому краю поля. Символ, що заповнює зайві позиції, можна змінити за допомогою маніпулятора setfill.

cout << «m=» << setiosflags(ios:: right) << setw(8) << m;

Напис «m=» буде вирівняно по лівому краю поля (за замовчанням), а значення змінної m буде вирівняно по правому краю поля шириною 8 позицій.

У потокових класах разом з операціями << і >> визначені методи для неформатованого читання і запису в потік. Наприклад:

– read(buf, num) – читає з потоку num символів (або всі символи до кінця файлу, якщо їх менше за num) в символьний масив buf і повертає посилання на поточний потік. Визначена у класі istream;

– write(buf, num) – записує в потік num символів з масиву buf і повертає посилання на потік. Визначена у класі ostream.

Запитання для самоконтролю

1. Які види потоків ви знаєте?
2. Для чого призначені стандартні потоки?
3. Який файл необхідно включати у програму для використання стандартних потоків?
4. Як називається операція «<<» ?
5. Як називається операція «>>» ?
6. Назвіть класи вхідних та вихідних потоків.
7. Який клас є базовим для потоків?
8. Для чого призначені маніпулятори потоків, і який файл необхідно включати у програму для їх використання?
9. Для чого призначені маніпулятори endl, setprecision, setw, setiosflags?

## *9.9 Файлові потоки вводу-виводу в С++*

У С++ кожен файл розглядається як послідовний потік байтів. За способом доступу файли можна розділити на послідовні, читання і запис в яких проводиться з початку байт за байтом, і файли з довільним доступом, що допускають читання і запис у вказану позицію. Файл завершується маркером кінця файлу (EOF – end-of-file marker).

Стандартна бібліотека містить три класи для роботи з файлами:

ifstream – клас вхідних файлових потоків;

ofstream – клас вихідних файлових потоків;

fstream – клас двонаправлених файлових потоків.

Використання файлів в програмі припускає наступні операції: створення потоку, відкриття потоку і скріплення його з файлом, обмін (введення/виведення), знищення потоку, закриття файлу.

Для використання файлів у С++ повинно включити заголовний файл <fstream>. Файл <fstream> включає визначення класів для створення файлових потоків: ifstream (для читання з файлу), ofstream (для запису у файл) і fstream (для читання і запису у файл). Ці класи потоків є похідними (тобто успадковують функціональні можливості) відповідно від класів ostream, iostream, istream. Таким чином, функции, операції (<< і >>) і маніпулятори можуть бути також застосовані і до потоків файлів.

Кожен клас файлових потоків містить конструктори, за допомогою яких можна створювати об'єкти цих класів різними способами. Конструктори без параметрів створюють об'єкт відповідного класу, не пов'язуючи його з файлом:

ifstream();

ofstream();

fstream().

Конструктори з параметрами створюють об'єкт відповідного класу, відкривають файл з вказаним ім'ям (*name) і пов'язують файл з вказаним об'єктом:

ifstream(const char *name, int mode = ios::in);

ofstream(const char *name, int mode = ios::out | ios::trunc);

fstream(const char *name, int mode = ios::in | ios::out).

Другим параметром конструктора є режим відкриття файлу. Якщо встановлене за замовчанням значення не влаштовує програміста, можна вказати інше (див. таблицю 9.2).

Відкрити файл в програмі можна з використанням або конструкторів, або методу open(), що має такі ж параметри, як і у відповідному конструкторі.

Для закриття потоку визначений метод close(), але оскільки він неявно виконується деструктором, явний виклик необхідний тільки тоді, коли потрібно закрити потік раніше кінця його області видимості.

Таблиця 9.2 – Режими відкриття файлів

| Режим | Опис |
| --- | --- |
| ios::app | Відкрити файл для додавання даних у кінець файлу |
| ios::ate | Перемістити покажчик в кінець файлу. |
| ios::in | Відкрити файл для читання (вводу). |
| ios::out | Відкрити файл для запису (виводу) |
| ios::trunc | Якщо файл існує, видалити (це також за замовчуванням робиться для ios::out). |
| ios::nocreate | Якщо файл не існує, то видати помилку. |
| ios::noreplace | Якщо файл існує, то видати помилку. |
| ios::binary | Відкрити файл у двійковому режимі |

Приклади відкриття файлу Tovar.dat для запису:

1) ofstream f(("Tovar.dat");    // оголошення файлового потоку виводу й відкриття файлу Tovar.dat для запису

2) ofstream f;    // оголошення файлового потоку виводу
f.open ("Tovar.dat");   // відкриття файлу Tovar.dat

3) fstream f ("Tovar.dat", ios::out);   // оголошення файлового потоку вводу/виводу й відкриття для запису

4) fstream f;    // оголошення файлового потоку вводу/виводу
f.open ("Tovar.dat", ios::out);    // відкриття файлу для запису

5) fstream f ("Tovar.dat", ios::app);   // оголошення файлового потоку вводу/виводу й відкриття файлу Tovar.dat для додавання

Приклади відкриття файлу Tovar.dat для читання:

1) ifstream f(("Tovar.dat");    // оголошення файлового потоку вводу й відкриття файлу Tovar.dat для читання

2) ifstream f;                    // оголошення файлового потоку вводу
   f.open ("Tovar.dat");   // відкриття файлу Tovar.dat

3) fstream f ("Tovar.dat", ios::in);       // оголошення файлового потоку вводу/виводу й відкриття для читання

4) fstream f;          // оголошення файлового потоку вводу/виводу
   f.open ("Tovar.dat", ios::in);          // відкриття файлу для читання

Після створення об'єкта будь-якого класу файлових потоків і спроби відкрити його, програма може перевірити, чи була операція відкриття файлу успішною. Наприклад:

```
if (!f)
{
    cerr «"Файл не може бути відкритий" << endl;  exit (1) ;
}
```

Деякі можливі помилки є слідством спроби відкрити для читання неіснуючий файл, спроби відкриття файлу для запису, коли на диску немає вільного місця і т.д. Коли умова вказує, що спроба відкриття файлу була безуспішною, виводиться повідомлення «Файл не може бути відкритий» і викликається функція exit для завершення програми. Аргумент функції exit повертається середовищу оточення, з якого програма була викликана. Аргумент 0 показує, що програма завершується нормально, а будь-яке інше значення, указує середовищу оточення, що програма припинила виконання із-за помилки. Значення, що повертає функція exit, використовується середовищем оточення (найчастіше операційною системою) для відповідної реакції на помилку.

Приклад запису у файл із використанням функції write (використано клас CTovar):

CTovar Tovar;      //оголошення об'єкта

ofstream fTovOut ("Tovar.dat", ios::app); // оголошення файлового потоку виводу fTovar і відкриття файлу для додавання

Tovar.input();                  //виклик функції input() об'єкта Tovar

fTovOut.write((char*)&Tovar,sizeof(Tovar));      // запис у файл даних об'єкта Tovar

fTovOut.close();             // закриття файлу

Приклад запису у файл із використанням операції << (помістити в потік):

```
void   CTovar::input()
 {
```

```
            ofstream fTovOut ("Tovar.dat", ios::app);
            cout <<"Введіть найменування товару ";
            cin >>name;
            cout <<"Введіть ціну товару ";
            cin >>price;
            fTovOut <<name <<' '<< price<<endl;      // запис у файл Tovar.dat
            fTovOut.close();
           }
```

Для читання з файлу можна використовувати функцію read() класу ifstream і операцію >> (узяти з потоку).

Приклад читання з файлу всіх даних з використанням функції read():

ifstream fTovIn("Tovar.dat"); // оголошення файлового потоку введення fTovar і відкриття файлу для читання

fTovIn.read ((char*)&Tovar,sizeof(Tovar));         // читання з файлу даних в об'єкт Tovar

```
while (fTovIn.Eof())               // поки не кінець файлу
{
   Tovar.output();
   fTovIn.read ((char*)&Tovar,sizeof(Tovar));
}
```

Приклад читання з файлу всіх даних з використанням операції >> (узяти з потоку) і вивід їх на екран:

```
void   CTovar::output()
 {
  ifstream fTovIn("Tovar.dat");
  cout <<"Найменування товару   "<<"  Ціна товару "<<endl ;
  while (fTovIn>>name>>price)        //поки є дані читати з файлу
  {
   cout<<"    "<<setiosflags(ios::left)<<setw(15) << name
         <<setprecision(2)     <<setiosflags(ios::right)<<setw(10)<<price<<"\n";
              }
 }
```

Запитання для самоконтролю

1. Які є файли за способом доступу?
2. Назвіть класи для створення файлових потоків і їх призначення.

3. Які дії необхідно виконати для використання файлів у програмі?
4. Які існують режими відкриття файлів?
5. Для чого призначений метод open() і які він має параметри?
6. Які способи відкриття та закриття файлу ви знаєте?
7. Як записати дані у файл?
8. Як прочитати дані з файлу?

## *9.10 Тести до розділу «Об'єктно-орієнтовне програмування»*

Відповіді та тести наведені у додатку А.

Тест 9.1. Клас – це:

а) стандартний тип даних;

б) тип даних, визначений користувачем;

в) об'єднання нестандартних типів даних;

г) спеціальний атрибут об'єкта.

Тест 9.2. Конструктор – це спеціальна функція, яка:

а) руйнує об'єкт;

б) конструює об'єкт;

в) ініціалізує об'єкт;

г) переміщує об'єкт.

Тест 9.3. Скільки конструкторів може бути оголошено в класі:

а) тільки один;

б) декілька;

в) жодного;

г) все вірно.

Тест 9.4. За замовчанням доступ до членів класу:

а) відкритий;

б) загальний;

в) закритий;

г) захищений.

Тест 9.5. Доступ до закритих членів класу мають:

а) тільки функції-члени даного класу;

б) функції-члени похідних класів;

в) функції-члени даного класу і друзі класу;

г) все вірно.

Тест 9.6. Клас може включати:

а) оголошення функцій;

б) оголошення даних;

в) оголошення доступу до членів класу;

г) все вірно.

Тест 9.7. Доступ до захищених членів класу мають:

а) функції-члени похідних класів;

б) функції-члени, будь-яких класів;

в) все невірно;

г) все вірно.

Тест 9.8. Знайдіть правильне оголошення конструктора класу Time:

а) void Time ();

б) int Time ();

в) Time ();

г) ~ Time ().

Тест 9.9. Знайдіть правильне оголошення деструктора класу Time:

а) void ~Time ();

б) ~int Time ();

в) Time ();

г) ~ Time ().

Тест 9.10. Дані-члени класу можуть приймати значення:

а) у оголошенні класу;

б) за допомогою деструктора;

в) за допомогою конструктора;

г) усе вірно.

Тест 9.11. На яких принципах засновано об'єктно-орієнтоване програмування:

а) інкапсуляція; успадкування; модульність; поліморфізм;

б) модульність; поліморфізм; інкапсуляція;

в) інкапсуляція; успадкування; модульність;

г) інкапсуляція; успадкування; поліморфізм.

Тест 9.12. Успадкування – це:

а) механізм, що дозволяє створювати нові класи ґрунтуючись на тих, що вже існують;

б) спосіб використання операцій одного класу в іншому класі;

в) повторення атрибутів одного класу в іншому класі;

г) немає правильної відповіді.

Тест 9.13. Поліморфізм означає:

а) можливість зміни об’єкта за бажанням користувача;

б) наявність безлічі форм або реалізацій конкретної функціональності;

в) максимально можливу незалежність кожного об’єкта;

г) немає правильної відповіді.

Тест 9.14. До яких елементів свого базового класу може мати доступ похідний клас 1) закритим 2) відкритим 3) захищеним:

а) 1) 2);

б) 1) 2) 3)

в) 2) 3)

г) 1) 3).

Тест 9.15. Конструктор викликається при:

а) створенні класу;

б) створенні об'єкта;

в) оголошенні класу.

Тест 9.16. Конструктор:

а) не може повертати значення;

б) може бути віртуальним;

в) може повертати значення типу void;

г) все вірно.

Тест 9.17. Що означає запис stud::set()?

а) функція stud знаходиться в області видимості класу set();

б) функція set() є конструктором;

в) функція set() знаходиться в області видимості класу stud.

Тест 9.18. Знайдіть правильне оголошення конструктора класу List:

а) void List ();

б) int List ();

в) ~ List ().

г) List ();

Тест 9.19. Як викликається конструктор?

а) відкрито вказав його ім’я;

б) автоматично при створенні екземпляра класу (об'єкта);

в) все вірно.

Тест 9.20. Інкапсуляція:

а) це незалежність внутрішньої структури об'єктів і прозорість їх інтерфейсів;

б) приховування даних і операцій від інших об'єктів;

в) можливість забезпечення спадкування властивостей об'єктів.

Тест 9.21. Для доступу до елементів класу використовуються операції:

a) .

b) :

c) ::

Тест 9.22. Атрибути класу – це:

a) всі елементи класу;

b) дані, визначені у функціях, які є членами класу;

c) дані у визначенні класу;

d) немає правильної відповіді.

Тест 9.23. Метод класу – це:

а) група інструкцій, що виконується як єдине ціле і призначена для виконання деяких дій;

б) група інструкцій для перетворення класу до іншого типу;

в) група даних, що визначена усередині класу.

Тест 9.24. Конструктор копіювання:

а) створює копію класу;

б) створює копію методів класу;

в) створює копію об’єкта будь-якого класу;

г) створює копію об’єкта цього ж класу.

Тест 9.25. Базовий клас:

а) це клас, з яким пов'язані всі класи;

б) це клас, в якому всі його члени автоматично стають членами похідних класів;

в) це клас, в якому зібрані всі атрибути як базового, так і породжуваного класів.

Тест 9.26. Абстрактний клас – це клас, який:

а) має хоч би одну віртуальну функцію;

б) має хоч би одну чисту функцію;

в) має хоч би одну чисту віртуальну функцію;

г) має тільки чисті віртуальні функції;

Тест 9.27. Запис fstream m ("f11.dat", ios::in) означає:

а) створення файлового потоку з ім’ям m для читання з файлу з ім’ям f11.dat;

б) створення файлового потоку з ім’ям m для запису даних у файл з ім’ям f11.dat;

в) створення об’єкта m класу ifstream для закриття файлу з ім’ям f11.dat.

Тест 9.28. Для запису даних у файл створюється об’єкт класу:

а) ifstream;

б) ofstream;

в) fstream.

Тест 9.29. Режим відкриття файлу для читання даних з нього:

а) ios::app

б) ios::out

в) ios::in

г) ios::binary

Тест 9.30. Запис class Bank_Dep: public Bank означає, що:

а) клас Bank є похідним від класу Bank_Dep;

б) клас Bank_Dep має доступ до відкритих членів класу Bank;

в) клас Bank_Dep є похідним від класу Bank;

г) правильної відповіді немає.

## СПИСОК ЛІТЕРАТУРИ


1. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения (Межгосударственный стандарт). URL: https://docs.cntd.ru/document/9041994 (дата звернення: 12.01.2022).
2. ГОСТ 19.003-80. Единая система программной документации. Схемы алгоритмов и программ. URL: https://www.swrit.ru/doc/espd/19.003-80.pdf (дата звернення: 01.12.2021).
3. Хьюз Дж., Мичтом Дж. Структурный подход к программированию. М. : Мир, 1980. 278 с. URL: https://www.twirpx.com/file/2196298/ (дата звернення: 01.12.2021).
4. Иодан Э. Структурное программирование и конструирование программ. М. : Мир, 1979. 416 с. URL: https://www.twirpx.com/file/1372924/ (дата звернення: 09.11.2021).
5. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. М. : Мир, 1981. 368 с. URL: https://www.twirpx.com/file/2499902/ (дата звернення: 07.10.2021).
6. Кузнєцов М.С. Процедурне програмування з використанням мови С: навч. посіб. Дніпропетровськ : НМетАУ, 2005. 84 с.
7. Шпак З.Я. Програмування мовою С : навч. посіб. Львів : Оріяна-Нова, 2006. 431 с.
8. Грицюк Ю.І., Рак Т.Є. Програмування мовою С++ : навч. посіб. Львів : Вид-во Львівського ДУ БЖД, 2011. 292 с. URL: http://xn--e1ajqk.kiev.ua/wp-content/uploads/2019/12/Griczyuk-C.pdf (дата звернення: 08.04.2022).
9. Жуковський С.С., Вакалюк Т.А. Об‘єктно-орінтоване програмування мовою С++ : навч.-метод. посіб. Житомир : Вид-во ЖДУ, 2016. 100 с. URL:https://dut.edu.ua/uploads/l_2138_51438193.pdf (дата звернення: 18.02.2022).
10. Бублик В.В. Об’єктно-орієнтоване програмування : підручник. К. : ІТкнига, 2015. 624 с. URL: https://library.kre.dp.ua/Books/2-4%20kurs/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%B8%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D

0%B2%D0%B0%D0%BD%D0%BD%D1%8F/bublik_vv_obiektnooriientovane_programuvannia-%D0%A1_2015.pdf (дата звернення: 10.05.2022).

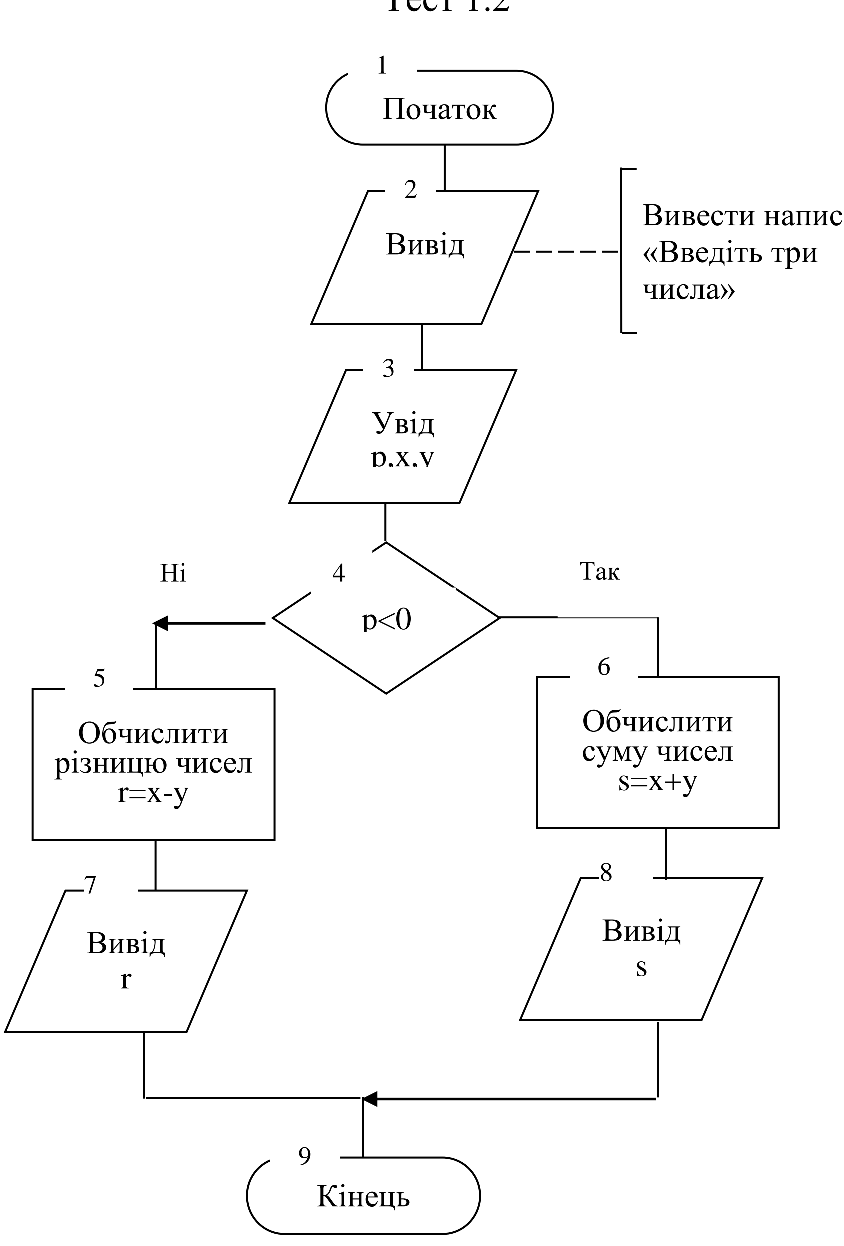
## Додаток А

### Відповіді на тести

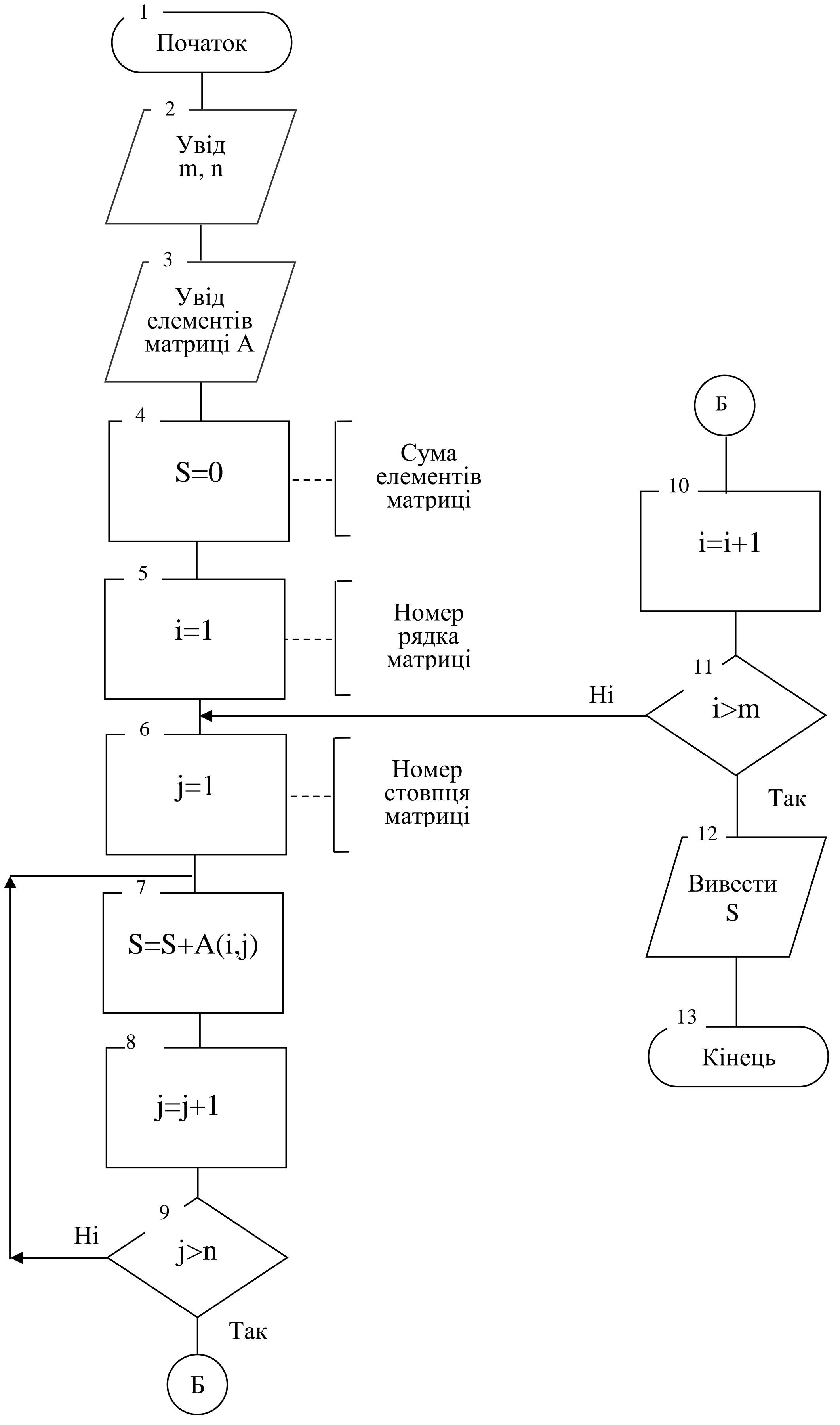
#### Тест 1.1

1) задати  $b$=15, $c$=20; $i$=1;
2) якщо ($i$ > 30),  то перейти до пункту 8;
3) увести з клавіатури $\mathbf{x_i}$;
4) обчистити $\mathbf{A_i}=\mathbf{b}*\mathbf{x_i}+\mathbf{c}$;
5) вивести на екран $\mathbf{A_i}$);
6) збільшити $i$ на одиницю;
7) перейти до пункту 2.
8) кінець.

#### Тест 1.2



Тест 1.3



Тест 1.4

Рядок 1: повинен містити пару круглих дужок вслід за ім'ям функції main, тобто main().

Рядок 2: повинна бути дужка {.

Рядок 3: правильний.

Рядок 4: a і s повинні розділятися комою.

Рядок 5: правильний.

Рядок 6: правильний, але змінній **b** ніде не присвоюється значення, тому **s** буде розраховане невідомо з яким значенням.

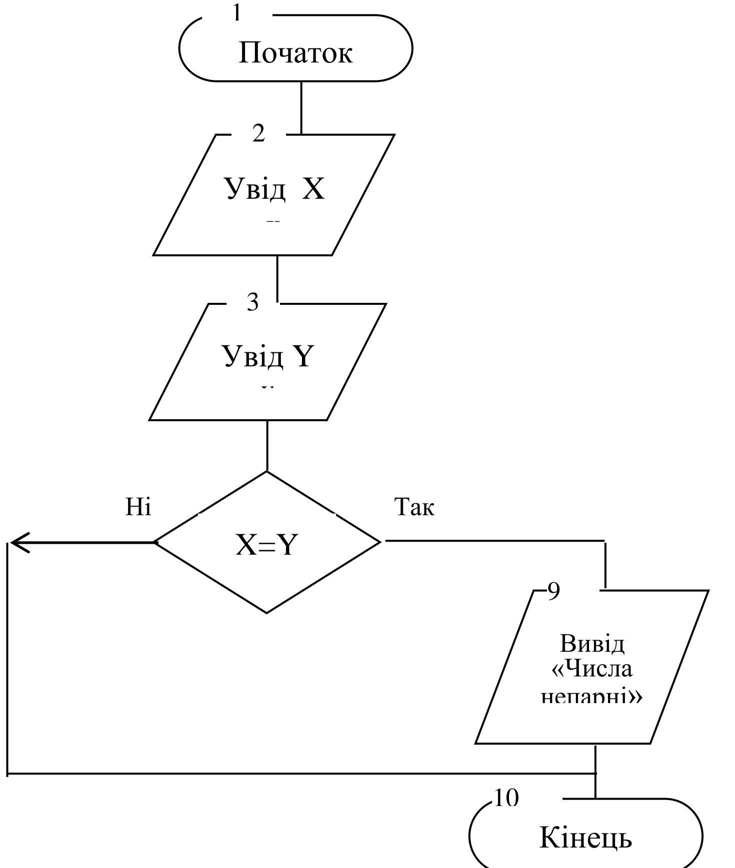
Рядок 7: повинна бути дужка {.

Тест 1.5

1) /34.8     /
2) /34.760000/
3) /  125/
4) /  0125/

де **/    /**– початок і кінець виводу.

Тест 2.1




```
#include "iostream"
using namespace std;

void main()
{
	int x,y;
	cout<<"x= ";
	cin >>x;
	cout<<"y= ";
	cin >>y;
	if (x==y)
	   cout <<"Числа рівні\n";
}
```

Тест 2.2

```
#include <iostream>
using namespace std;
void main()
{
	int x, k, y=0, z=0;
	setlocale (0, "");		// Коректне відображення кирилиці
	cout<< "Введіть x:";
	cin >> x;
	if(x>-5 && x<=5)  {
		y=2*x+7;
		cout <<"\n y="<< y;
	}
	cout <<"\nВведіть k:";
	cin >> k;
	if(k<-10 || k>10)    {
		z=3*k;
		cout <<"\n z=" << z;
	}
	if(y<z)  cout <<"\ny  менше  ніж z\n";
		else cout <<"\nz  менше ніж y\n";
	system ("pause");
}
```

Тест 2.3.

a=5  y=25  k=30

Тест 3.1

```
#include <windows.h>
#include "iostream"
using namespace std;
void main()
{
	char stat;
	int kol_m=0, kol_g=0;
	SetConsoleCP(1251);		// для вводу кирилиці
	SetConsoleOutputCP(1251);	// для виводу кирилиці
	for (int i=1;i<=10;i++)  {
		cout << "\nВведіть стать (ч/ж)";
```

```
            cin >> stat;
            switch(stat)   {
              case 'ч': kol_m++;       break;
              case 'ж': kol_g++;       break;
                     default: cout << "\nНевідома стать";
            }
      }
  cout << "\nКількість жінок -"<< kol_m  << "кількість чоловіків -" <<
kol_g;
}
```

Тест 3.2

1) Програма з оператором ***for***

```
#include "iostream"
using namespace std;
void  main()  {
    int numbe,  i, n, sum=0;
    cout <<"Введіть кількість чисел \n";
    cin >> n;
    cout << "Введіть  числа \n";
    for (i=1; i<=n; i++)  {
        cin >> numbe;
        sum=sum+numbe;
    }
    cout <<"Сума " <<n << "чисел дорівнює "<<sum;
}
```

2) Фрагмент програми з ***while***

```
i=1;
while (i<=n)
{
   cin >> numbe;
   sum=sum+numbe;
   i++;
}
```

3) Фрагмент програми з ***do while***

```
i=1;
do {
   cin >> numbe;
   sum=sum+numbe;
   i++;
} while (i<=n);
```

Тест 3.3

```
#include "iostream"
using namespace std;
void  main()  {
      float x,y;
      for (x=-1; x<=1; x+=0.2)  {
            y=3*x-4;
            cout << "\nx=" << x <<"\t y=" << y;
      }
}
```

Тест 4.1

```
#include "iostream"
using namespace std;
int square (int);               // функція піднесення до квадрату
void main()  {
      int i=20;
      while(i<=100)  {
            cout <<"\n", square(i);
            i++;
      }
}
int square (int x)
{
   return(x*x);
}
```

Тест 5.1

```
#include "iostream"
using namespace std;
void main() {
     int i, j, a[10][10], m, n;
     cout <<"\n Введіть число рядків";
     cin <<m;
     cout << "Введіть число стовпців";
     cin  << n;
     for (i=0; i<m; i++)
          for (j=0; j<n; j++)
               cin >> a[i][j];
```

```
    for (i=0; i<m; i++) {
        for (j=0; j<n; j++)
            cout <<a[i][j] << "\t";
        cout << "\n";
    }
}
```

Відповіді на тести до розділу «Об’єктно-орієнтоване програмування»

| Номер тесту | 9.1 | 9.2 | 9.3 | 9.4 | 9.5 | 9.6 | 9.7 | 9.8 | 9.9 | 9.10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Відповідь | **б** | **в** | **г** | **в** | **в** | **г** | **а** | **в** | **г** | **в** |
| Номер тесту | 9.11 | 9.12 | 9.13 | 9.14 | 9.15 | 9.16 | 9.17 | 9.18 | 9.19 | 9.20 |
| Відповідь | **г** | **а** | **б** | **в** | **б** | **а** | **в** | **г** | **б** | **а** |
| Номер тесту | 9.21 | 9.22 | 9.23 | 9.24 | 9.25 | 9.26 | 9.27 | 9.28 | 9.28 | 9.30 |
| Відповідь | **а** | **с** | **а** | **г** | **б** | **в** | **а** | **б** | **в** | **в** |

Навчальне видання

Бандоріна Лілія Миколаївна

Климкович Тетяна Олександрівна

Удачіна Катерина Олександрівна

**ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ**

Навчальний посібник

Тем. план. 2022, поз 246

Підписано до друку 18.04.2022. Формат 60х84 1/16. Папір друк. Друк плоский. Облік.-вид. арк. 9,29. Умов. друк. арк. 9,17. Замовлення № 42.

Навчально-науковий інститут
«Інститут промислових та бізнес технологій»
Українського державного університету науки і технологій
49005, м. Дніпро, пр. Гагаріна, 4
Редакційно-видавничий відділ ННІ ІПБТ УДУНТ