МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

О.С. Зеленський

В.С. Лисенко

ОСНОВИ ПРОГРАМУВАННЯ

на C++

Навчальний посібник

Кривий Ріг

2023

Навчальний посібник «Основи програмування на С++» призначений для поглибленого вивчення основ класичного програмування на мові С++. В якості основ розглядається структурне програмування, яке є базисом для подальшого вивчення об’єктно-орієнтованого програмування. Навчальний посібник адресований студентам, слухачам магістратури, аспірантам, викладачам. Може бути використаний як самовчитель.

Автори: Зеленський О.С., Лисенко В.С. – Кривий Ріг: Державний університет економіки і технологій, 2023.-269 с.

Рецензенти:

А.І. Купін, д.т.н, професор, завідувач кафедри комп’ютерних систем та мереж, Криворізький національний університет.

І.О. Музика, к.т.н, доцент кафедри комп’ютерних систем та мереж, декан факультету інформаційних технологій Криворізький національний університет.

С.В. Баран, к.е.н., доцент кафедри інформатики і прикладного програмного забезпечення, Державний університет економіки і технологій.

Рекомендовано Вченою радою Державного університету економіки і технологій

Протокол № 9 від 23.02.2023 р.

ЗМІСТ

ВСТУП

Навчальний посібник «Основи програмування на С++» призначений для поглибленого вивчення студентами основ класичного програмування на мові С++. В якості основ розглядається структурне програмування, яке є базисом для подальшого вивчення об'єктно-орієнтованого програмування. На конкретних прикладах розглянуті алгоритми та програми по основним розділам структурного програмування. У кожному розділі наводиться реалізація прикладів та виведення відповідних результатів. Значна увага приділяється структурам даних, обробці виключних ситуацій, родовим функціям, масивам та роботі з вказівками. У окремих розділах приведені типові алгоритми сортування даних, а також чисельні методи вирішення задач (чисельне диференціювання та інтегрування, чисельне вирішення рівнянь та систем рівнянь).

При реалізації прикладів мають місце декілька варіантів виконання певних операцій, на яких треба зупинитися. У цьому випадку всі варіанти окрім першого коментуються (символи коментарю "//" або "/*…*/").

Зміст навчального посібника охоплює наступні розділи:

1. Арифметичні основи обчислювальних машин.
2. Основи алгоритмізації обчислювальних процесів.
3. Основні відомості С++.
4. Введення-виведення даних.
5. Операції в С++.
6. Організація циклів.
7. Зовнішні пристрої та символьне введення/виведення. Рядкові, числові функції та функції роботи з датою та часом.
8. Вказівки, посилання та масиви.
9. Робота з функціями.
10. Рекурсивне програмування.
11. Робота з файлами.
12. Структури.
13. Об'єднання та інші типи даних. Обробка виключних ситуацій.
14. Динамічні структури даних.
15. Типові методи сортування масивів.
16. Чисельне диференціювання та інтегрування.
17. Чисельні методи розв'язання алгебраїчних рівнянь.
18. Чисельні методи розв'язання системи лінійних алгебраїчних рівнянь.

Приклади, приведені в посібнику розроблені для режиму Console (тип

проекту Win32 Console Application) з використанням сучасного стандарту C++ (ISO/IEC 14882). Даний стандарт застосовується у мовах програмування Visual C++, починаючи з 1998 року. У Visual C++ 2002-2022 років підтримується тільки сучасний стандарт і лише в Visual C++ 1998 року підтримується попередній та сучасний стандарти програмування.

Відмінності між попереднім і сучасним стандартами програмування включають дві нові риси: змінився стиль оформлення заголовків (headers) і з'явилася інструкція namespace.

Заголовки (headers), які пов’язані з потоками введення/виведення даних пишуться без розширення h, всі ж інші заголовки залишаються без змін. Це такі хайдери, як <iostream>, <fstream>, <istream>, <ostream>, <iomanip> та інші хайдери, пов’язані з потоками. Коли в програму включається заголовок нового стилю, зміст цього заголовка знаходиться в просторі імен std. Простір імен (namespace) – це оголошувана область, необхідна для того, щоб уникнути конфліктів імен ідентифікаторів. Щоб простір імен std став видимим, необхідно використовувати наступну інструкцію: using namespace std.

Навчальний посібник адресований студентам, слухачам магістратури, аспірантам, викладачам. Може бути використаний як самовчитель.

РОЗДІЛ 1. АРИФМЕТИЧНІ ОСНОВИ ОБЧИСЛЮВАЛЬНИХ МАШИН

1.1. Позиційні системи числення. Переведення числа з однієї системи числення в іншу

Числа, що оброблюються технічними засобами, в першу чергу електронно-обчислювальною машиною (ЕОМ), представляються в них у вигляді спеціальних кодів в прийнятій системі числення.

ЕОМ – це комплекс апаратних, архітектурних та програмних засобів, призначених для автоматичної обробки інформації.

Система числення є сукупністю прийомів позначення (запису) чисел за допомогою знаків. Розрізняють непозиційні та позиційні системи числення. В непозиційних системах числення значення кожного знаку не має чіткої прямої залежності від його положення в числі.

Сучасна позиційна система числення виникла в Індії в V столітті н.е. В позиційній системі числення значення кожної цифри залежить від її позиції (розташування) в числі.

При зображенні чисел в позиційній системі числення розрізняють **базу** і **основу** системи.

База – це послідовна сукупність знаків, за допомогою яких записуються числа. Кількість цифр, за допомогою яких записуються числа у даній системі числення, називається основою системи числення. Основа показує у скільки разів одиниця наступного (лівого) розряду більше одиниці попереднього (правого) розряду.

Будь-яке число у позиційній системі числення можна представити у вигляді ряду:

$$N = \sum_{n=k}^{n=-m} a_n q^n = a_k q^k + a_{k-1} q^{k-1} + ... + a_1 q^1 + a_0 q^0 + a_{-1} q^{-1} + a_{-2} q^{-2} + ... + a_{-m} q^{-m},$$

де q – основа системи числення; n – номер розряду; k та m – цілі додатні числа-номери старшого та молодшого розрядів; a_n – цілі додатні числа від 0 до q-1 ($0 \le a_n \le q-1$), що показують скільки одиниць n-го розряду міститься у числі.

Для людини природнім є запис чисел у десятковій системі числення. Наприклад:

$$123.12 \rightarrow 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 1 \cdot 10^{-1} + 2 \cdot 10^{-2},$$

$$a_{n(10)} = \{0;1;2;3;4;5;6;7;8;9\}.$$

Для ЕОМ десяткова система числення не найзручніша. Оскільки для представлення окремих розрядів необхідно мати фізичні елементи, що мають десять різноманітних стійких станів, що дуже важко. Ще складніше реалізувати апаратно виконання арифметичних операцій над даними числами.

Застосування електричної енергії та винахід електронних елементів, що мають два стійких стани (є потенціал або імпульс або нема потенціалу або імпульсу), обумовили широке застосування двійкової системи числення. Будь-яке число в цій системі числення можна позначити за допомогою двох цифр: 1 або 0

$a_{n(2)} = \{0;1\}$.

Окрім десяткової системи числення при обробці інформації також використовуються 8-а та 16-а, а також двійково-десятковий код. 8-а та 16-а системи числення застосовуються для скорочення довжини запису при кодуванні програми та розміщенні даних в пам'яті ЕОМ.

$a_{n(8)} = \{0;1;2;3;4;5;6;7\}$.

$a_{n(16)} = \{0;1;2;3;4;5;6;7;8;9;A;B;C;D;E;F\}$.

У якості проміжку між десятковим та двійковим записом використовується двійково-десятковий код. Ряд машин застосовують цей код лише при введенні даних, переводячи його потім у двійкову систему. У таблиці 1.1 представлені числа від 0 до 17 у 10-й, 2-й, 8-й та 16-й системах числення.

Розглянемо переведення чисел із однієї системи числення в іншу.

Таблиця 1.1

Ряд чисел у 10-й, 2-й, 8-й та 16-й системах числення

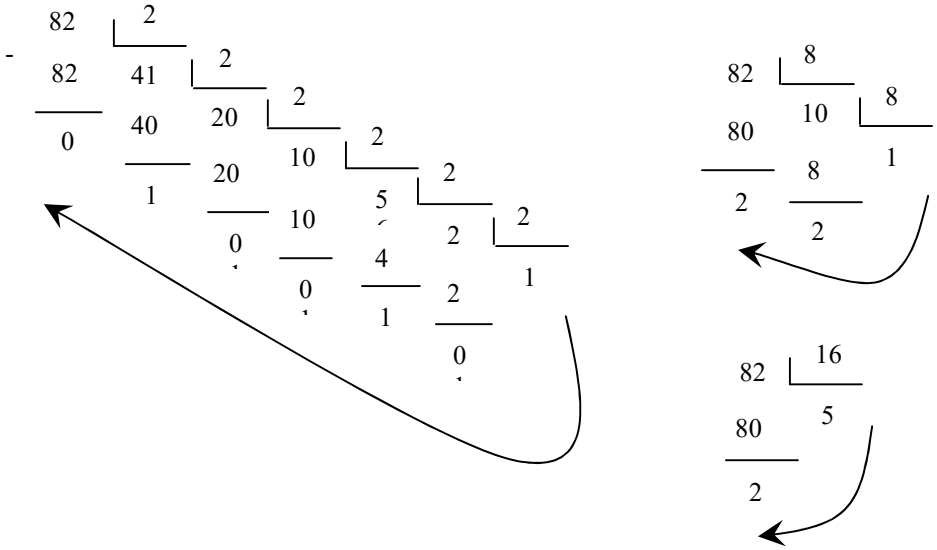
q=10	q=2	q=8	q=16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11

Для переведення цілого десятинного числа в систему числення з основою q необхідно поетапно ділити його на основу нової системи числення, поки не отримаємо остаток менше за q. Число в новій системі числення записується у вигляді залишків від ділення на кожному етапі починаючи з останнього.

Наприклад, потрібно перевести число $82_{(10)}$ в 2-у, 8-у та 16-у систему числення:

$82_{(10)} = ?_{(2)} = ?_{(8)} = ?_{(16)}$.

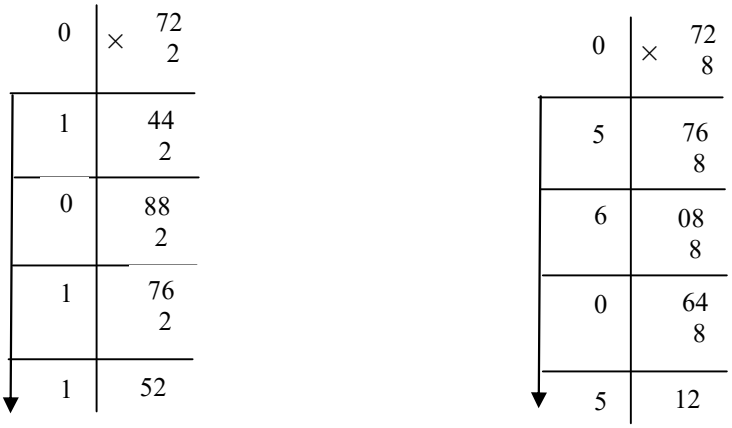
Для переведення десятинних дробів в систему числення з основою q необхідно послідовно помножити дріб на цю основу і виділяти цілі частини числа при кожній операції множення. Множення закінчується, коли черговий добуток стає рівним нулю або при отриманні заданого ступеню точності.



$82_{(10)} = 1010010_{(2)} = 122_{(8)} = 52_{(16)}$

Число в новій системі числення представляється в вигляді послідовності цифр цілих частин добутку.

Наприклад, розглянемо переведення числа $0.72_{(10)} = S_{(8)} = S_{(2)}$ з точністю до чотирьох знаків після коми:



$0.72_{(10)} = 0.1011_{(2)} = 0.5605_{(8)}$

$82.72_{(10)} = 1010010.1011_{(2)} = 122.5605_{(8)}$

Переведення числа із системи числення з основою q в десятинну виконується наступним чином:

$1010010.1011_{(2)} = 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} +$

$+ 1 \cdot 2^{-4} = 82.6875_{(10)}$

Для переведення з 2-ї в 8-у або 16-у систему числення та навпаки користуються спрощеним прийомом в зв'язку з тим, що числа 8 та 16

відповідно дорівнюють 2^3 та 2^4. Для переведення з 2-ї в 8-у (16-у) систему числення достатньо починаючи з десятинної точки цілу та дробову частину числа розбити на тріади (тетради) і кожну тріаду (тетраду) замінити 8-ю (16-ю) цифрою. Переведення 8-х та 16-х чисел в 2-у систему числення здійснюється в зворотному порядку.

Наприклад:

$\underbrace{10}_{2}\underbrace{01}_{3}\underbrace{11}_{5}\underbrace{01}_{4}.\underbrace{100}_{4}$ **(2)**

2 3 5 4 **(8)**

$\underbrace{1001}_{9}\underbrace{1101}_{D}\underbrace{1000}_{8}$ **(2)**

9 D 8 **(16)**

Тетрада – 4 розряди

Тріада – 3 розряди

1.2. Форма представлення чисел. Кодування чисел

Сукупність розрядів регістру, призначених для зберігання чисел, називається розрядною сіткою машини. В розрядній сітці числа можуть бути представлені в природній або нормальній формі. Природна форма інакше називається формою представлення числа з фіксованою точкою, а нормальна – з плаваючою точкою.

Форма представлення чисел з фіксованою точкою

Ця форма використовується для запису цілих чисел. Ціле число може займати 1, 2, 4 байти. 1 байт – 8 біт – 8 розрядів. Форма запису цілого числа в комірку, довжиною в 1 байт:

зн							

зн – 1, якщо число < 0

0, якщо число > 0.

Від'ємні числа можуть бути представлені в прямому, зворотному та додатковому кодах.

В прямому коді достатньо в знаковий розряд помістити 1.

В зворотному коді всі нулі заміняються на 1, а 1 на нулі.

Додатковий код формується таким же чином як і зворотний, але додається 1 в молодший розряд.

Приклади:

0	0	0	0	0	1	0	1	+5

1	0	0	0	0	1	0	1	-5	Прямий код

1	1	1	1	1	0	1	0	-5	Зворотний код

1	1	1	1	1	0	1	1	-5	Додатковий код

Використання додаткового та зворотного коду дозволяє операцію віднімання звести до операції додавання, а множення до операції додавання та здвигу.

Фома представлення чисел з плаваючою точкою

В нормальній формі будь-яке число складається з двох груп

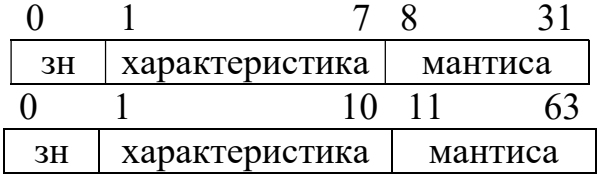
$$N = \pm mq^{\pm p}$$

де m – мантиса, тобто правильний дріб <1;

p – порядок числа;

q – основа системи числення

Числа з плаваючою точкою представляються в двох форматах: короткому – 4 байти (32 біти) та довгому – 8 байт (64 біти).



У випадку комірки довжиною 4 байти точність складає 7 десятинних значущих чисел. З використанням комірки 8 байт точність збільшується до 16 десятинних значущих розрядів.

Наведемо приклад запису числа 178.125, як 4-х байтової змінної з плаваючою точкою.

На першому етапі потрібно перевести задане число до двійкової системи числення (відповідно, його цілої та дробової частини).

$178.125_{(10)} = 10110010.001_{(2)}$.

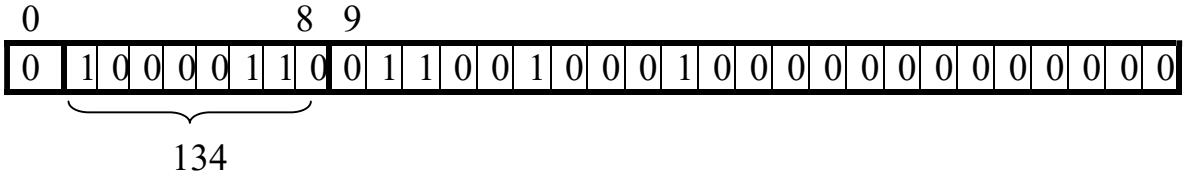
На другому етапі необхідно його нормалізувати. Стандартом IEEE 754 передбачений наступний механізм нормалізації, за яким у цілій частині числа залишається цифра 1. Так як у двійковій системі числення, при нормалізації числа, на першому місці буде завжди залишатись 1-ця, це дає змогу зекономити 1 біт та не записувати цю одиницю до мантиси. Таким чином, цей біт використовується на розширення порядку числа, а кількість бітів мантиси зменшується на один. Отже, для запису числа з плаваючою точкою розміром в 4 байти, фактично на порядок числа відводиться 7+1 = 8 біт, а на мантису 24–1 = 23 біта (але число в мантису записується розміром в 24 біта, так як 1-ця не пишеться, але має місце).

При нормалізації число $10110010.001_{(2)}$ буде виглядати наступним чином: $1.0110010001*2^7$.

На третьому етапі переведемо порядок числа в двійкову систему числення. Так як порядок розширюється на 1 біт – він може приймати значення від 0 до 255. Але треба врахувати від'ємний та додатній порядок. Нульове значення порядку приймається рівним 127 (2^7–1). Отже, від'ємний порядок – від 0 до 126, 127 – характеризує нульове значення порядку (база), від 128 до 255 – додатній порядок. В нашому випадку порядок дорівнює 7, тобто 127 + 7 = $134_{(10)}$.

$134_{(10)} = 10000110_{(2)}$

На останньому етапі запишемо отримане число в комірку пам'яті.



Стандарт IEEE 754 застосовується сучасними засобами програмного забезпечення низького та високого рівнів на сучасних персональних комп'ютерах.

Діапазони чисел в 4-байтовій та 8-байтовій комірках:

$-2^{128}<X<2^{128}$ або

$-3.402823*10^{38}<X<3.402823*10^{38}$ (для 4-х байтів);

$-2^{1024}<X<2^{1024}$ або

$-1.79769313486231*10^{308}<X<1.79769313486231*10^{308}$ (для 8-ми).

1.3. Арифметичні операції над двійковими числами. Машинні одиниці інформації

Арифметичні дії виконуються у відповідності з таблицями додавання та множення.

Додавання	Множення
0 + 0 = 0	0 * 0 = 0
1 + 0 = 1	1 * 0 = 0
0 + 1 = 1	0 * 1 = 0
1 + 1 = 10	1 * 1 = 1

Наприклад:

+ 5 – 5 = 0

	101	+ 5
	11111011	- 5
1	00000000	0

→ відкидається

$$\begin{array}{r} 111 \\ \times\ 101 \\ \hline 111 \\ \quad 111 \\ \hline 100011 \end{array}$$

Операція множення зводиться до операцій додавання та здвигу.

Контрольні питання

1. Що таке позиційна система числення і як знайти значення числа за його записом у певній позиційній системі?
2. Як перевести десяткове число до будь-якої іншої системи числення?
3. Як перевести число з будь-якої системи числення до десяткової?
4. Вкажіть основні форми зображення чисел у комп'ютері.
5. Яким чином здійснити переведення чисел між 2-ю, 8-ю та 16-ю системами числення?
6. Яким чином представити додатне число з фіксованою точкою у

пам’яті комп’ютера?

7. Яким чином представити від’ємне число з фіксованою точкою у пам’яті комп’ютера?

8. Яким чином представити число з плаваючою точкою у пам’яті комп’ютера?

9. Наведіть нормальну форму запису числа з плаваючою точкою.

10. Яким чином відбувається додавання, віднімання та множення у двійковій системі числення?

Завдання

Метою даної роботи є представлення чисел з фіксованою та плаваючою точкою, а також взаємні переведення чисел з однієї системи числення в інші – між 2-ю, 8-ю, 10-ю, та 16-ю системами числення. Дана робота є базовою по темі 1 "Арифметичні основи обчислювальних машин" даного посібника. Особлива увага приділяється стандарту IEEE 754, який застосовується сучасними засобами програмного забезпечення низького та високого рівнів на сучасних персональних комп’ютерах.

Варіанти індивідуальних завдань

1. Перевести число $5196.2_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
2. Перевести число $6285.4_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($6285_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
3. Перевести число $7374.6_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
4. Перевести число $8463.8_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($8463_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
5. Перевести число $9542.9_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
6. Перевести число $8651.7_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($8651_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
7. Перевести число $7732.5_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
8. Перевести число $6823.3_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($6823_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
9. Перевести число $5914.1_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
10. Перевести число $6805.6_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($6805_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну

з фіксованою точкою.

11. Перевести число $7716.2_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
12. Перевести число $8627.4_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($8627_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
13. Перевести число $9538.6_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
14. Перевести число $8749.8_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($8749_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
15. Перевести число $7358.9_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
16. Перевести число $6267.7_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($6267_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
17. Перевести число $5176.5_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
18. Перевести число $4085.4_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($4085_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
19. Перевести число $3194.3_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
20. Перевести число $2283.1_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($2283_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
21. Перевести число $1372.6_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
22. Перевести число $2461.2_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($2461_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
23. Перевести число $3550.4_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.
24. Перевести число $6642.6_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати цілу частину числа ($6642_{(10)}$) у комірку пам’яті ЕОМ, як 2-х байтову змінну з фіксованою точкою.
25. Перевести число $5733.7_{(10)}$ в 2-у, 8-у та 16-у системи числення. Записати число у комірку пам’яті ЕОМ, як 4-х байтову змінну з плаваючою точкою.

РОЗДІЛ 2. ОСНОВИ АЛГОРИТМІЗАЦІЇ ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ

2.1. Поняття і властивості алгоритму

Алгоритм – це організована послідовність точно визначених дій, необхідних для розв'язання поставленої задачі.

Ефективним методом побудови алгоритмів є метод покрокової деталізації, при якому завдання розбивається на кілька простих під задач (модулів), і для кожного модуля створюється свій власний алгоритм.

Основні властивості алгоритму:

1. Дискретність – процес розв'язку розбивається на окремі етапи, а етапи на окремі кроки, виконання яких зводиться до виконання елементарних операцій – додавання, віднімання та ін.
2. Визначеність (точність) – чіткість вказівок, які утворюють алгоритм, їх повна зрозумілість та однозначність, тобто опис будь-якої дії у алгоритмі не повинен бути розтлумачений по-різному будь-якими двома людьми.
3. Зрозумілість – усі дії, включені до алгоритму, мають бути у межах компетенції виконавця алгоритму.
4. Універсальність (масовість) – алгоритм має бути придатний до розв'язання множини однотипних задач, має виконуватись при довільних вхідних даних та початкових умовах.
5. Скінченність – алгоритм має бути реалізований за визначене число кроків і повинен працювати з визначеними вхідними даними.
6. Результативність – при будь-яких значеннях вхідних даних результат повинен досягатися за визначений час при виконанні визначеної кількості кроків.

Алгоритми – це не тільки опис послідовності рішення різноманітних задач. У формі різноманітних інструкцій та правил алгоритми супроводжують людину протягом усього життя. Добираючись від дому до роботи або місця навчання, кожен з нас виконує певний алгоритм, який ми склали, виходячи з власних можливостей, умов та накопиченого досвіду. В залежності від часу виходу з дому, погодних умов, інших обставин цей алгоритм може допускати декілька варіантів маршрутів, використання того чи іншого виду транспорту тощо.

2.2. Засоби представлення алгоритмів

Існують наступні засоби представлення алгоритмів:

1. Словесний.
2. Формульно-словесний.
3. Графічний (у вигляді блок-схем).
4. Операторний.
5. З використанням алгоритмічної мови.

6. Табличний.

Прикладом словесної форми завдання алгоритму є алгоритм Евкліда для знаходження спільного дільника двох чисел (a та b):

1. Оглядаючи два числа a та b переходьте до наступного пункту.
2. Порівняйте ці два числа (a дорівнює b, a менше b, a більше b).
3. Якщо a та b рівні, то припиніть обчислення: кожне з чисел дає шуканий результат. Якщо числа не рівні переходьте до наступного пункту.
4. Якщо перше число менше другого, то поміняйте їх місцями.
5. Віднімайте друге число із першого, оглядаючи два числа: від'ємник та залишок, після чого переходьте до п. 2.

За цим алгоритмом можна знайти найбільший спільний дільник для двох натуральних чисел.

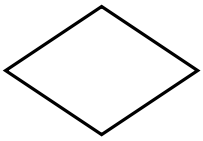
Найбільш широко розповсюдженим є графічний спосіб опису алгоритму. Цей спосіб є найбільш доступним, швидко засвоюється та в значній мірі формалізований. Алгоритм описується у вигляді блок-схем, що представляють графічне зображення процесу вирішення задачі. Кожний з блоків має певне значення та відображає деякий етап вирішення задачі. У блоці рідко вказуються елементарні операції; найважливіше тут – це їх логічна послідовність і розгалуження.

Алгоритми програм відображаються у графічній формі у вигляді схем за допомогою умовних графічних позначень – символів згідно «ГОСТ 19.701 – 90 (ИСО 5807-85). Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Издательство стандартов, 1991».

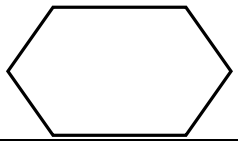
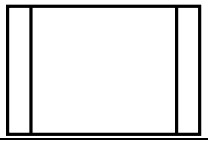
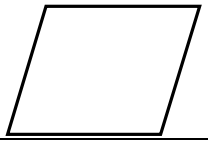
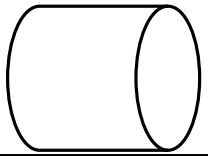
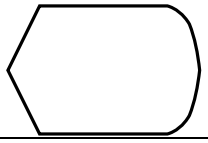
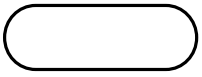
Умовні зображення блоків та їх функції наведені у таблиці 2.1.

Таблиця 2.1

Блоки, які найчастіше використовуються у схемах алгоритмів

Назва	**Графічне зображення**	**Опис**
Процес		Відображає функцію обробки даних будь-якого вигляду (виконання певної операції або групи операцій, що приводить до зміни значення, форми або розміщення інформації або до визначення, по якому з декількох напрямів потоку слід рухатися)
Рішення		Відображає рішення або функцію перемикача типу, що має один вхід і ряд альтернативних виходів, один і лише один з яких може бути активізований після обчислення умов, визначених усередині цього символу. Відповідні результати обчислення можуть бути записані по сусідству з лініями, що відображають ці шляхи.

Продовження таблиці 2.1

Назва	Графічне зображення	Опис
Модифікація		Відображає модифікацію команди або групи команд з метою дії на деяку подальшу функцію (установка перемикача, модифікація індексного регістра або ініціалізація програми).
Зумовлений процес		Відображає зумовлений процес, що складається з однієї або декількох операцій або кроків програми, які визначені у іншому місці (у підпрограмі, модулі).
Введення-виведення		Відображає перетворення даних у форму, придатну для обробки (введення) або відображення результатів обробки (виведення).
Документ		Відображає дані, представлені на носієві в легкій для читання формі (машинограма, документ для оптичного або магнітного читання, мікрофільм, рулон стрічки з підсумковими даними, бланки введення даних).
Пристрій з прямим доступом		Відображає дані, які зберігаються, в запам'ятовуючому пристрої з прямим доступом (магнітний диск, магнітний барабан, гнучкий магнітний диск).
Дисплей		Відображає введення-виведення даних, якщо безпосередньо підключений до процесора пристрій відтворює дані і дозволяє операторові ЕОМ вносити зміни в процес їх обробки.
Канал зв'язку		Відображає передачу даних по каналу зв'язку.
Лінія потоку		Відображає потік даних або управління.
З'єднувач		Відображає вихід в частину схеми і вхід з іншої частини цієї схеми та використовується для обриву лінії і продовження її у іншому місці. Відповідні символи-з'єднувачі повинні містити одне і те ж унікальне позначення.
Термінатор		Відображає вихід в зовнішнє середовище і вхід із зовнішнього середовища (початок або кінець схеми програми, зовнішнє використання і джерело або пункт призначення даних).
Коментар		Символ використовують для додавання описових коментарів або записів пояснень. Пунктирні лінії в символі коментарю пов’язані з відповідним символом або можуть обводити групу символів.

При складанні алгоритмів вирішення економічних задач окрім графічного способу широко використовується табличний спосіб запису алгоритмів. В табличній формі звичайно описуються машинні документи.

2.3. Типи алгоритмічних процесів. Приклади

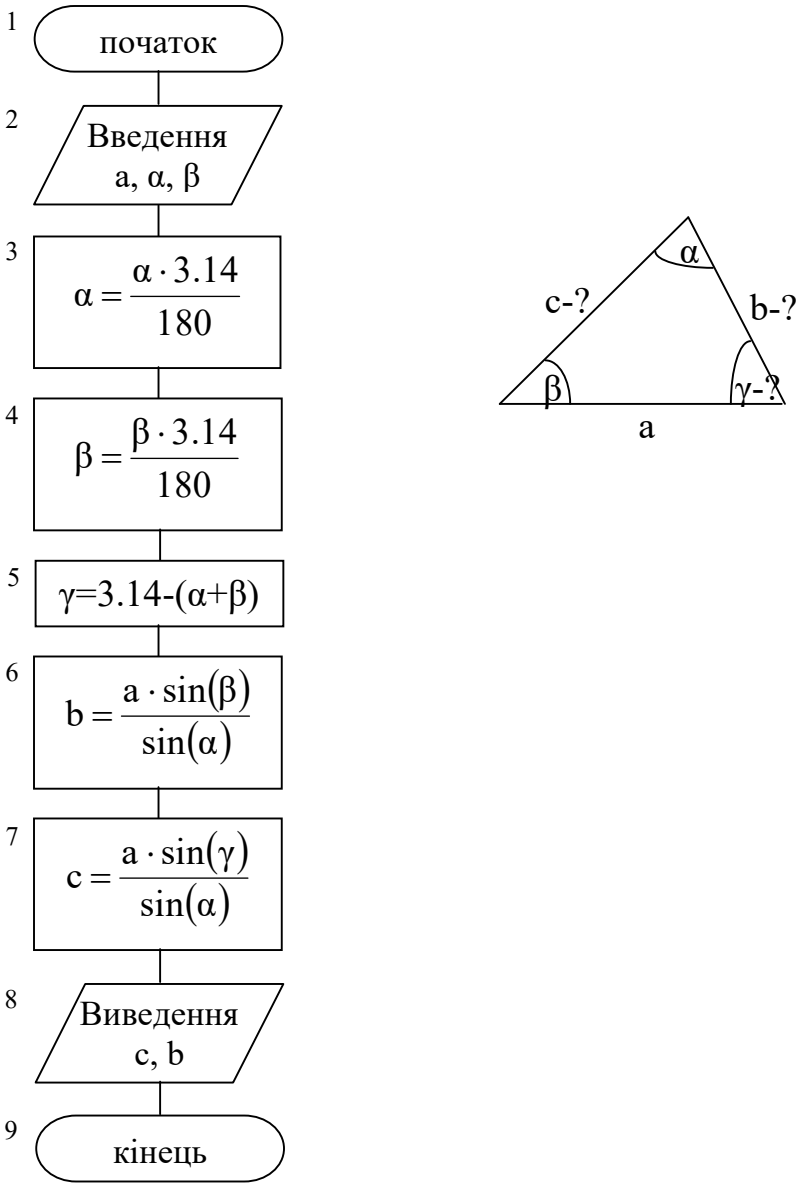
Усі алгоритмічні процеси можна віднести до трьох основних типів:

1. Лінійний.
2. Розгалужений.
3. Циклічний.

Лінійні обчислювальні процеси

Послідовний (лінійний) обчислювальний процес характеризується тим, що кроки алгоритмів виконуються строго послідовно у тому порядку, в якому вони представлені.

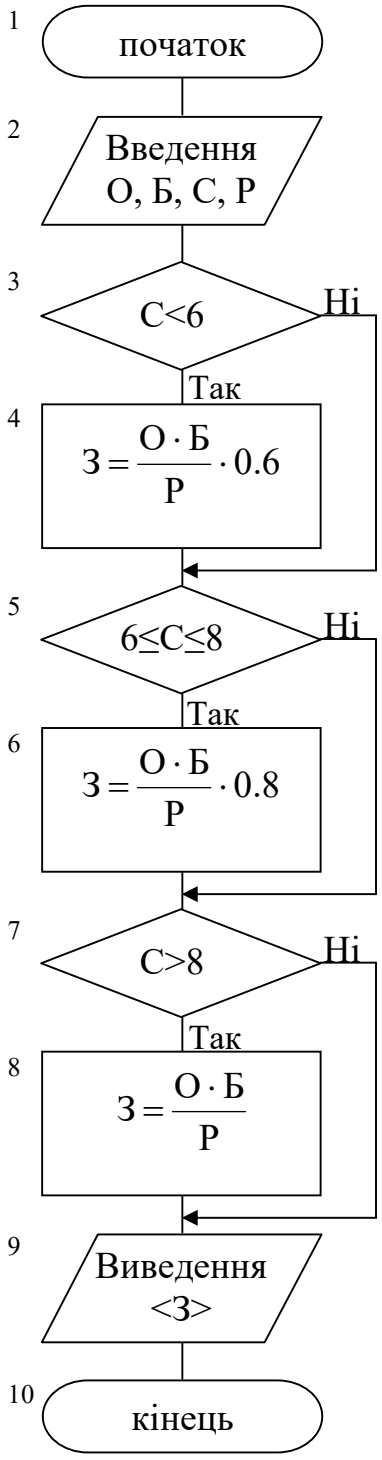
Приклад. Скласти блок-схему обчислення довжин двох сторін трикутника b, c по відомій довжині сторони a і двом внутрішнім кутам α і β.

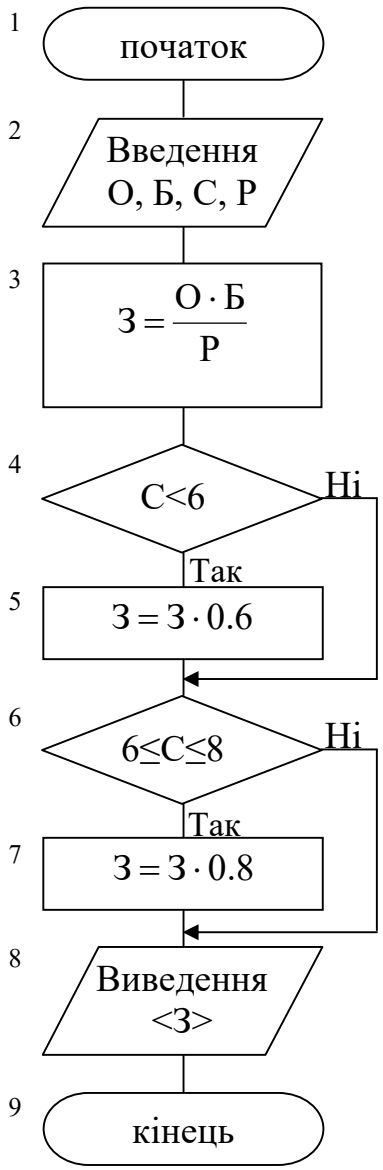


Розгалужені обчислювальні процеси

Обчислювальні процеси, в яких в залежності від значення деякої ознаки відбуваються обчислення по одному з деяких можливих спрямувань, називають розгалуженими.

Розглянемо гіпотетичний приклад. Для робітника зі стажем роботи менше 6 років оплата за лікарняним листом складає 60% від окладу; від 6 до 8 років – 80%; більше 8 років – 100%. Необхідно скласти блок-схему з нарахування оплати по лікарняному листу. Означення: О – оклад; Б – кількість днів хвороби; С – стаж; З – нарахування по лікарняному листку; Р – кількість робочих днів.

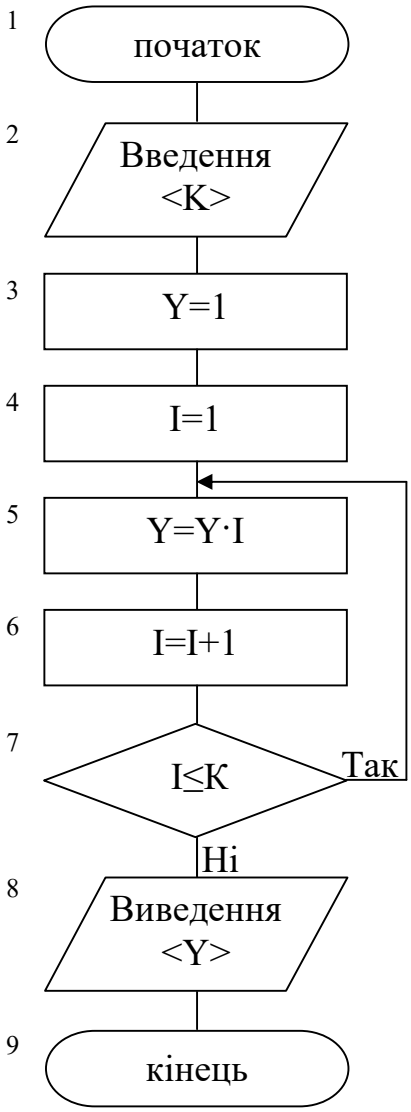


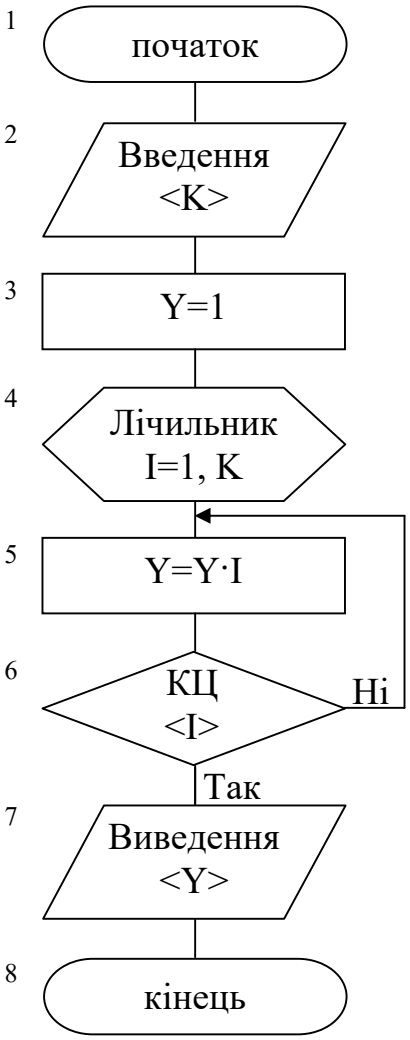


Циклічні обчислювальні процеси

Етап (ділянка) алгоритму, який у процесі вирішення задачі виконується багато разів, називається циклом. Відповідно, обчислювальні процеси, які містять цикли називають циклічними. Цикли бувають арифметичні та ітераційні. На відміну від ітераційних в арифметичних циклах відома кількість ітерацій.

Приклад. Обчислити: y=k!

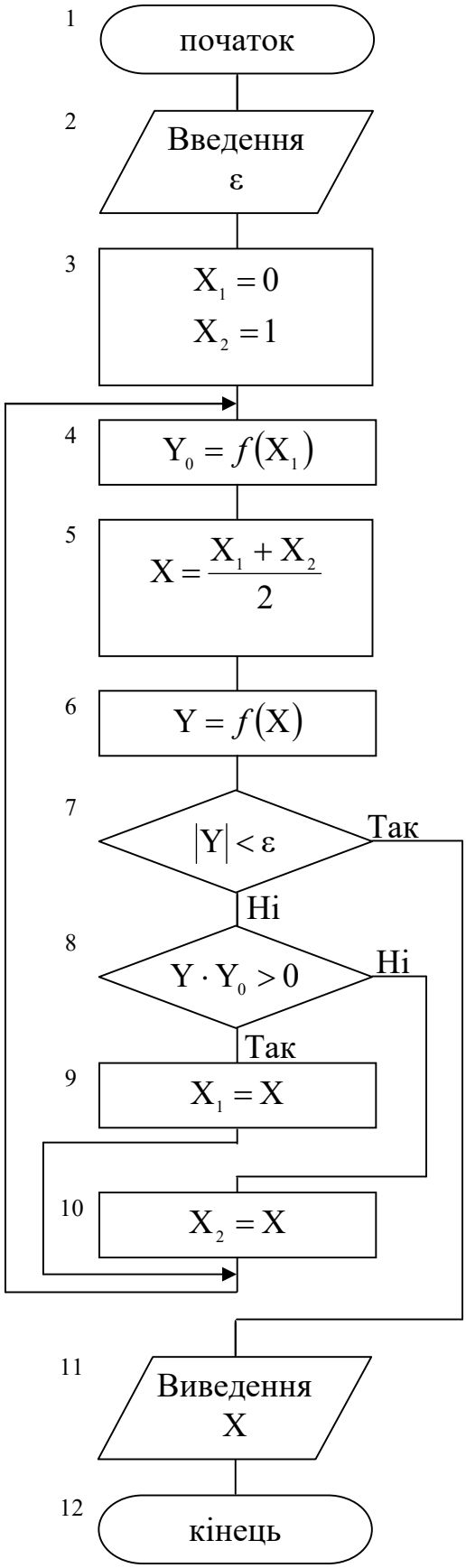




Приклад ітераційних циклів.

Задано функцію: $y = x^2 + e^x - 2$

Знайти x, який знаходиться в діапазоні від 0 до 1, при якому $y \approx 0$.


1
початок
2
Введення
ε
3
$X_1 = 0$
$X_2 = 1$
4
$Y_0 = f(X_1)$
5
$X = \frac{X_1 + X_2}{2}$
6
$Y = f(X)$
7
$|Y| < \varepsilon$
Так
Ні
8
$Y \cdot Y_0 > 0$
Ні
Так
9
$X_1 = X$
10
$X_2 = X$
11
Виведення
X
12
кінець

Контрольні питання

1. Що таке алгоритм?
2. Перелічить основні етапи та кроки виконання алгоритму.
3. Наведіть властивості алгоритму.
4. Які існують засоби представлення алгоритмів?
5. Наведіть типи алгоритмічних процесів.
6. Наведіть приклад лінійного обчислювального процесу.
7. Наведіть приклад розгалуженого обчислювального процесу.
8. Наведіть приклад циклічного обчислювального процесу (арифметичний та ітераційний цикли).
9. Чим відрізняються арифметичний та ітераційний циклічні процеси?
10. Наведіть основні умовні зображення блоків представлення алгоритму.

Завдання

Завдання по кожному з алгоритмічних процесів будуть даватися поступово, по мірі проходження необхідних конструкцій мови програмування C++.

РОЗДІЛ 3. ОСНОВНІ ВІДОМОСТІ С++

3.1. Загальна характеристика програми

Програма – це набір інструментів, за допомогою яких комп'ютер працює з даними. Мова програмування дозволяє створювати ці інструменти. Спочатку програмування виконувалося з використанням команд, що розуміються комп'ютером безпосередньо. Це повільний й утомливий процес, так що програмісти стали створювати мови програмування високого рівня, якими було простіше користуватися. Такі мови не могли сприйматися машиною прямо, тому були створені спеціальні програми, які назвали компіляторами, для перекладу цих нових мов у систему команд комп'ютера. С++ є середовищем для розробки програм, що містить у собі як компілятор, так і деякі інші інструменти. На виході компілятора утворюється файл із розширенням **obj**, це розширення позначає об'єкт. Використовуваний файл є файлом, що містить відкомпільовану і готову до виконання програму. Іноді комп'ютерні файли, що виконуються називають завантажувальним модулем або просто програмою.

Оператори мов низького рівня дуже схожі на систему команд самого комп'ютера. Ці програми можуть бути дуже ефективними. Вони важкі для сприйняття, декілька рядків тексту можуть виконувати таку операцію як друк символу. Вони специфічні для конкретного процесора та мають сильну залежність між мовою і комп'ютером. Мови низького рівня часто називають мовами асемблера.

Мови високого рівня набагато зручніше для сприйняття людиною. Написати таку програму простіше. Вони практично не залежать від апаратної платформи. Багато мов високого рівня сертифіковані міжнародними організаціями, такими як ANSI (Американський національний інститут стандартів). Така стандартизація робить мову переносною. Приклади: FORTRAN, COBOL, PASCAL, С++.

Програма на мові С++ складається з функцій, описів та директив препроцесору. Гловна функція програми повинна мати ім'я **main**. Виконання програми починається з першого оператору цієї функції. Просте визначення функції має наступний формат:

```
тип_повертаємого_значення ім'я ( [ параметри ])
{
оператори, які складають тіло функції
}
```

Як правило, функція використовується для обчислення якого-небудь значення, тому перед ім'ям функції вказується її тип. Перелічимо необхідні відомості по функціям:

– якщо функція не повинна повертати значення, вказується тип **void**;

– тіло функції є блоком і, отже, береться у фігурні дужки;

– кожен оператор закінчується крапкою з комою (окрім складеного оператору).

Приклад структури програми, яка містить функції main, f1 та f2:

```
директиви препроцесора
описи функцій

void main()
{
 оператори головної функції
}
int f1()
{
 оператори функції f1
} .
int f2()
{
 оператори функції f2
}
```

Програма може складатися з декількох модулів. Кожний модуль представлений файлом з розширенням cpp.

3.2. Сучасний стандарт C++

Традиційна версія C++ базується на розробці Бьярна Страуструпа. Друга версія C++ створена Бьярном Страуструпом спільно з комітетом із стандартизації (ANSI – American National Standards Institute, Американський національний інститут стандартів; ISO – International Standards Organization, Міжнародна організація по стандартах) є надбудовою традиційного C++ (стандарт ISO/IEC 14882).

Відмінності між попереднім і сучасним стилями програмування включають дві нові риси: змінився стиль оформлення заголовків (headers) і з'явилася інструкція namespace. Для демонстрації цих відмінностей розглянемо дві версії простої програми на C++. Перша версія написана в попередньому традиційному стилі програмування.

```
#include <iostream.h>
void main()
{
 // програмний код
}
```

Інструкція **#include** підключає до програми заголовний файл **iostream.h**, який забезпечує підтримку системи введення/виведення C++.

Нижче представимо другу версію програми, в якій використовується сучасний стиль:

```
#include <iostream>
using namespace std;
void main()
{
 // програмний код
}
```

В перших двох рядках на початку програми мають місце зміни. По-перше в інструкції **#include** після слова **iostream** відсутні символи **.h**. По-друге, в наступному рядку задається так званий простір імен (**namespace**).

Оскільки нові заголовки є іменами файлів без розширення, для них не потрібно указувати розширення **.h**, а тільки ім'я заголовка в кутових дужках. Нижче представлені заголовки, які підтримуються в сучасному стандарті мови С++:

```
<iostream>
<fstream>
<istream>
<ostream>
<iomanip>
<vector>
<string>
```

Передусім для нового стандарту змінені заголовки для роботи з потоками – це заголовки, що закінчуються на “stream”, або починаються на “io”.

Такі заголовки як і раніше включаються в програму за допомогою інструкції **#include**. Єдиною відмінністю є те, що нові заголовки пишуться без розширення.

Оскільки С++ містить всю бібліотеку функцій С, як і раніше підтримується стандартний стиль оформлення заголовних файлів бібліотеки С. Таким чином, такі заголовні файли, як **stdio.h** і **ctype.h** все ще доступні. Проте сучасний стандарт С++ також визначає заголовки нового стилю, які можна указувати замість цих заголовних файлів. Відповідно версії С++ до стандартних заголовків С просто додається префікс **c** та видаляється розширення **.h**. Наприклад, заголовок **math.h** замінюється новим заголовком С++ **<cmath>**, а заголовок **string.h** – заголовком **<cstring>**. Всі компілятори С++ підтримують заголовки старого стилю. Проте такі заголовки оголошені застарілими і не рекомендуються.

Заголовок **<string.h>** (**<cstring>**) включає в себе функції роботи з рядками (масивами символів). Його не слід плутати с заголовком **<string>**, який призначений для роботи з класом **string** та має місце при вивченні об’єктно-орієнтованого програмування.

Коли в програму включається заголовок нового стилю, зміст цього заголовку знаходиться в просторі імен **std**. Простір імен (namespace) – це оголошувана область, необхідна для того, щоб уникнути конфліктів імен ідентифікаторів. Щоб простір імен **std** став видимим, необхідно

використовувати наступну інструкцію:

```
using namespace std;
```

Ця інструкція поміщає **std** в глобальний простір імен. Після того, як компілятор обробить цю інструкцію, можна працювати із заголовками як старого, так і нового стилю.

У Visual C++ 2002-2022 років підтримуються для роботи з потоками заголовки лише нового стилю (<**iostream**>, <**fstream**>, <**ostream**>, <**istream**>, <**iomanip**>). Інші ж заголовні файли, такі як <**math.h**>, <**string.h**> мають місце, але рекомендується використовувати заголовки нового стилю, відповідно <**cmath**>, <**cstring**>. У подальших прикладах будуть використовуватись заголовки тільки сучасного стандарту.

3.3. Кроки для створення та виконання програми

Кроки для створення та виконання програми продемонстровані на рис.3.1.

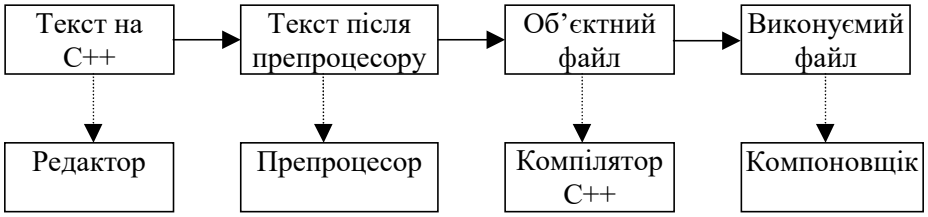


Рис. 3.1. Процес створення та виконання програми

Розглянемо програму **First.cpp**.

```
// Файл First. cpp
// Програма на C++ для починаючих
// демонструє коментар і показує
// декілька оголошень і представлень

#include <iostream>

using namespace std;

void main()
{
  int i, j;char c;
  double x;
  i=4;j=i+7;
  c='A';     //Символьні константи містяться
             //в одиночні лапки
  x= 9.087; //Плаваюча точка
  x=x*4.5;
  cout<< i << j << c << x;
}
```

Програма має вбудовані ключові слова, наприклад: **int, char, return**. По суті справи С++ складається з набору функцій (функціонально закінчених частин коду) і функції **main()**. Весь код між фігурними дужками називають блоком. У нашому прикладі функція **main()** має один блок.

Оператори повинні закінчуватися “;”. Рядки з **main()** і фігурними дужками не закінчуються “**;**”.

Коментарі використовуються в такий спосіб. Якщо коментар займає 1 рядок, то – “//”. Якщо коментар займає більш 1 рядку, то початок коментарю виділяють символами “/*”, а кінець “*/”.

Основні арифметичні операції:

\+ - * /

3.4. Змінні та константи

Змінні характеризуються атрибутами:

- адреса в пам’яті;
- ім’я;
- тип;
- значення.

Першою буквою змінної повинна бути буква латинського алфавіту, або знак підкреслення (_), потім букви і цифри або додаткові знаки підкреслення.

Заголовні і малі літери в змінній мають різноманітний зміст.

У таблиці 3.1. розглянемо типи змінних.

Таблиця 3.1

Типи змінних мови програмування С++

Ім’я оголошення	**Тип**	**Кількість байтів**	**Діапазон**	**Значність**
char	Символьний	1	-128 - 127	–
unsigned char	Беззнаковий символьний	1	0 - 255	–
signed char	Символьний	1	-128 - 127	–
wchar_t	Розширений символьний	2	-32768 - 32767	
int	Цілий	4	-2147483648-2147483647	–
unsigned int	Беззнаковй цілий	4	0 - 4294967295	–
signed int	Цілий	4	-2147483648-2147483647	–
short int	Короткий цілий	2	-32768 - 32767	–
unsigned short int	Беззнаковий короткий цілий	2	0 - 65535	–
signed short int	Короткий цілий	2	-32768 - 32767	–
long int	Довгий цілий	4	-2147483648-2147483647	–
signed long int	Довгий цілий	4	-2147483648-2147483647	–

Продовження таблиці 3.1.

Ім'я оголошення	Тип	Кількість байтів	Діапазон	Значність
unsigned long int	Беззнаковий довгий цілий	4	0 - 4294967295	–
bool	Логічний	1	true (1), false(0)	–
float	З плаваючою точкою.	4	-3.4E+38 - 3.4E+38	7
double	Довгий із плаваючою точкою	8	-1.7E+308 - 1.7E+308	16
long double	Довгий із плаваючою точкою (має місце лише в Borland C++, у Visual C++ – аналог типу даних double)	10	-1.1E+4932 – 1.1E+4932	18

Звернемо увагу на типи даних, які з'явились у новому стандарті C++: **wchar_t** (розширений символьний), **bool** (логічний).

Тип **wchar_t** призначений для роботи з набором символів, для кодування яких недостатньо 1 байта, наприклад, Unicode. Розмір цього типу залежить від реалізації. Як правило, він відповідає типу **short**. Рядкові константи типу **wchar_t** записуються з префіксом L, наприклад, L"Gates".

Величини логічного типу **bool** можуть набувати значень **true** і **false**, що є зарезервованими словами. Внутрішня форма представлення значення **false** – 0 (ноль). Будь-яке інше значення інтерпретується як **true**. При перетворенні до цілого типу **true** має значення 1.

Тип може бути перевизначений:

```
typedef int CARDINAL;
CARDINAL pp;
```

У C++ цілі та символьні змінні можуть зберігати чисельні значення. Для присвоювання значень змінній використовується знак "=".

Змінна = < вираз >

Можна одночасно оголошувати та привласнювати змінні:

```
int age = 30;
char first = 'G', m = 'H';
```

Константи.

Цілі константи не мають десяткових точок; числа з плаваючою точкою (дробові) повинні мати десяткову точку.

Цілі константи (з фіксованою точкою) мають по умовчанню тип даних int. Можна явно вказати тип цілої змінної за допомогою суфіксів L, 1 (long) та u, U (unsigned). Наприклад, константа 32L буде мати тип long та займати 4 байта. Можна використовувати суфікси L та U одночасно, наприклад, 22UL или 5Lu.

Цілі константи можуть задаватися 8-ми та 16-ми константами – відповідно префікси 0 та 0x.

Приклад:

```
05, 07, 0x10; 0x2C4; 0xFFFF
unsigned short int t1 = 0177777;  //65535
unsigned short int t2 = 0xFFFF;   //65535
short int t3 = 0xFFFF;            //-1
short int t4 = 0177777;           //-1
```

Константи з плаваючою точкою мають по умовчанню тип double. Для того щоб явно вказати тип константи float, необхідно використати суфікси F, f. Наприклад, константа 5.43f буде мати тип даних float.

Рядкові константи.

Рядкові константи містяться у лапки, наприклад:

```
"C++ Програма",   "123".
```

Усі рядкові константи закінчуються ASCII-нулем. Він є обмеженням рядка в пам’яті. Не треба плутати ASCII-нуль із звичайним нулем, що має код 48. Наприклад, рядок "A760" у пам’яті ЕОМ має вид:

'A'	01000001
'7'	00000111
'6'	00000110
'0'	00110000
'\n'	00000000

Довжина рядкової константи визначається числом символів без ASCII-нуля.

Символьні константи.

Символьні константи повинні бути замкнуті в одинарні лапки.

При запровадженні спеціальних символів (Escape-послідовності) вимагаються спеціальні позначення, що можна представити наступною таблицею (табл. 3.2.).

Таблиця 3.2

Символи Escape-послідовності

Керуюча послідовність	**Значення**
\a	Звуковий сигнал
\b	Повернення на одну позицію
\f	Переведення сторінки (для принтера)
\n	Новий рядок (переведення рядка, вертання каретки)
\r	Вертання каретки
\t	Табуляція
\\	Ліва похила риса
\?	Знак питання

Продовження таблиці 3.2.

Керуюча послідовність	Значення
\'	Одиночні лапки
\"	Подвійні лапки
\0	ASCII – нуль

Приклади

```
char c;
c = 'T'+5; //T - ASCII - 84
cout << c; //Y - ASCII - 89
cout << "Привіт"<< '\n'<< "Вася";
```

```
Привіт
Вася
```

```
#include <iostream>
using namespace std;

void main()
{
char bell = '\a';
cout<<bell; // Звук
}
```

3.5. Директиви препроцесора

Директиви препроцесора – це команди для препроцесора. Вони не є командами С++. Починаються зі знаку #, наприкінці “ **;** ” не ставиться Директиви препроцесора використовуються до компіляції програми.

Приклад:

```
#include <iostream>
#define age 28
#define message "Привіт!"
```

Директива **#include** використовує 2 формати:

```
#include <файл>
#include "файл"
```

Кутові дужки вказують препроцесору шукати файл, який включається у зазначених каталогах (у нашому випадку в зазначеному каталозі IDE). Звичайно це каталог **INCLUDE**. Якщо зазначені (**“ ”**), то пошук файла здійснюється в каталозі, де знаходиться вихідний текст, а потім, якщо він не буде знайдений, то в тих же каталогах, що й у випадку з кутовими дужками.

Препроцесор цілком вставляє текст зазначеного файла замість директиви **#include**.

Директива #include частіше усього використовується для включення заголовних файлів, (**h**-файли). У цих файлах описується, як інтерпретувати бібліотечні функції, використовувані в програмі. Наприклад: **iostream** – містить необхідну для компілятора інформацію про бібліотечні оператори потокового введення-виведення **cout, cin**, а також про інші бібліотечні функції, що забезпечують введення - виведення.

При підключенні заголовних файлів стандартної бібліотеки розширення **.h** можна не вказувати. Це зроблено для того, щоб не обмежувати способи їх зберігання. Для кожного файлу бібліотеки С з ім'ям **<name. h >** є відповідний файл бібліотеки С++ **<cname>**, в якому ті ж засоби описуються в просторі імен **std**. Наприклад, директива **#inclucle <cstdio>** забезпечує ті ж можливості, що і **#include <stdio.h>**, але при зверненні до стандартних функцій потрібно указувати ім'я простору імен **std**.

Призначення директиви **#define** – знайти та замінити, наприклад:

```
#define DD 28
```

В наступному прикладі по всьому тексту перед компіляцією препроцесор змінює x1 на b+c.

Приклад:

```
#include <iostream>
using namespace std;

#define x1 (b+c)
#define x2 (x1+x1)

void main()
{
 int b=2;
 int c= 3;
 int e = x2; //b+c+b+c
 cout <<e; //10
}
```

Щоб не плутати аргументи **#define** із змінними у програмі, їх виділяють заголовними буквами. В основному **#define** використовується для визначення констант. Альтернативно, оголошення констант може відбуватися так:

```
const float c=10.2f;
const char d = 'A';
```

Ці константи видимі тільки усередині блоку { }.

Таким чином, директива **#define** є чудовим засобом для визначення цифрових та рядкових констант.

Директиви умовної компіляції

Директиви умовної компіляції #if, #ifdef та #ifndef застосовуються для того, щоб виключити компіляцію окремих частин програми. Це буває корисно при відладці або, наприклад, за підтримки декількох версій програми для різних платформ.

Формат директиви #if:

```
#if константное_выражение
[ #elif константное_выражение
[ #elif константное_выражение
.]

[ #else
.]
#endif
```

Кількість директив #elif – довільне. Блоки коду, що виключаються, можуть містити як описи, так і виконувані оператори. Приклад умовного включення різних версій заголовного файлу:

```
#if VERSION == 1
#define INCFILE "vers1.h"
#elif VERSION == 2
#define INCFILE "vers2.h" / * і так далі * /
#else
#define INCFILE "versN.h"
#endif
#include INCFILE
```

У константних виразах може використовуватися перевірка, чи визначена константа, за допомогою defined(имя_константы), наприклад:

```
#if defined(__BORLANDC___)& & ___BORLANDC_ == 0x530 // Bc5.3:
typedef istream_iterator <int.char.char__traits<char>.ptrdiff__t>
istream_iter;
#elif defined(_BORLAND___)/ / BC5.2:
typedef istream_iterator<int. ptrdiff__t> istream_iter:
#else // VC5.0:
typedef  istream_iterator<int> istream_iter:
#endif
```

Інше призначення директиви – тимчасово закоментувати фрагменти коду, наприклад:

```
#if 0
int i, j ;
double x, y;
#endif
```

Оскільки допускається вкладеність директив, такий спосіб дуже зручний.

Найчастіше в програмах використовуються директиви #ifdef та #ifndef,

що дозволяють управляти компіляцією залежно від того, чи визначений за допомогою директиви #define вказаний в них символ (хоч би як порожній рядок, наприклад, #define 32_BIT_SUPPORT):

```
#ifdef символ
//Розташований нижче код компілюється,
//якщо символ визначений
#ifndef символ
//Розташований нижче код компілюється,
//якщо символ не визначений
```

Дія цих директив розповсюджується до першого #elif, #else або #endif.

Директива #ifndef часто застосовується для того, щоб забезпечити включення заголовного файлу тільки один раз:

```
#ifndef HEADER_INCLUDED
#include "myheader.h"
#define HEADER_INCLUDED
#endif
```

Контрольні питання

1. Що таке програма?
2. Опишіть процес створення програми.
3. Які стандартні типи даних є у мові C++?
4. Які відмінності існують між сучасним стандартом C++ та його попереднім стандартом?
5. Які типи констант ви знаєте?
6. Які директиви препроцесора ви знаєте?
7. Призначення директиви #include.
8. Призначення директиви #define.
9. Перелічите символи Escape-послідовності.
10. Яким чином можна перевизначити тип даних.
11. Яким чином можна задати 8-і та 16-і константи?
12. Рядкові та символьні константи.
13. Які ви знаєте засоби ініціалізації масиву символів (рядка)?
14. Розкрийте призначення ASCII-нуля ‘\0’?

РОЗДІЛ 4. ВВЕДЕННЯ-ВИВЕДЕННЯ ДАНИХ

4.1. Об'єкт виведення даних cout

Об'єкт **cout** виводить дані на заданий пристрій.

Синтаксис:

cout << data[<< data],

де **data** – змінні, константи, вирази.

Приклад:

```
cout << "Привіт";
cout << "Коля"; //ПривітКоля

cout<< "\n Привіт\n";
cout << "Коля";
//Привіт
//Коля

# include <iostream>
using namespace std;

void main()
{
cout << "Петро \tВася\n";
cout << "3\t2\n";
cout << "4\t5\n";
}
```

Виведення:

```
Петро Вася
3       2
4       5
```

\t - табуляція через 8 символів.

Для **cout** використовуються різноманітні маніпулятори. Наприклад, маніпулятори **hex** і **oct** використовуються для виведення відповідно, 16-х та 8-х чисел.

Приклад:

```
# include <iostream>
using namespace std;

void main()
{
 int num = 0x4c, num_2 = 012, num_s;
 num_s = num + num_2;
```

```
    cout << "Виведення у 16-х кодах " << hex << num_s;
    //Виведення у 16-х кодах 56
}
```

Маніпулятор установлює виведення 8 та 16–річних чисел, поки не зустрінеться інший маніпулятор **oct, hex, dec**.

Маніпулятори **setw(), setprecision()** і **setfill()** – це функції-члени які змінюють стан об'єкта **cout**.

setw() – вирівнює число виведення в межах заданої ширини.

setfill() – заповнює незаповнені позиції встановленим символом. При цьому формуються пробіли зліва.

setprecision(n) – маніпулятор виведення зазначеної кількості знаків **n**.

Приклад:

```
#include <iostream>
#include <iomanip>
using namespace std;

void main()
{
cout << 456 <<456 <<456<<"\n";
//456456456
cout << setw(5) << 456 << setw(5) << 456<< 456;
//  456 456456
cout<<"\n"<<setw(10)<<setprecision(6)<<12.47888;
//   12.4789
cout << "\n"<< setw(10) << setfill('+')<< 12.47;
//+++++12.47
}
```

Маніпулятори **setiosflags** і **resetiosflags** використовуються для установки певних глобальних прапорів, які клас C++ **iostream** використовує при визначенні поведінки по умовчанню при введенні та виведенні. На ці прапори посилаються як на ***змінні стану***. Функція **setiosflags()** встановлює зазначені в ній прапори, а **resetiosflags()** очищує (або скидає) їх. Ці маніпулятори в якості аргументів використовують значення, приведені в таблиці 4.1.

Таблиця 4.1

Аргументи для setiosflags і resetiosflags

Значення	Результат, якщо значення встановлене
ios::skipws	Ігнорує порожній простір при введенні
ios::left	Виведення із вирівнюванням зліва
ios::right	Виведення із вирівнюванням справа
ios::dec	Виведення у десятковому форматі
ios::oct	Виведення у восьмеричному форматі
ios::hex	Виведення у шістнадцятирічному форматі

Продовження таблиці 4.1.

Значення	Результат, якщо значення встановлене
ios::showbase	Виводити основу системи числення
ios::showpoint	Виводити десяткову точку
ios::suppercase	Виводити шістнадцятирічне число заголовними буквами
ios::showpos	Виводити “+” перед додатними цілими числами
ios::scientific	Використовувати наукову форму виведення чисел із плаваючою точкою
ios::fixed	Використовувати форму виведення чисел із фіксованою точкою

Наприклад:

```
cout<<setw(10)<<"Привіт\n";
cout<<setw(10)<<setiosflags(ios::left)<<"Привіт";

Результат виконання:

    Привіт
Привіт
```

4.2. Функція виведення даних printf

Для виведення даних у класичному С використовується функція **printf.** Вона без змін перенесена у С++.

Синтаксис:

printf(char *fotmat, <додаткові аргументи>)

Рядок формату складається із символів виведених без зміни та спеціальних символів формату, які задають перетворення даних. Для такого перетворення використовуються додаткові аргументи.

Специфікація перетворення формату має вигляд:

%[прапори] [ширина] [. точність] [L|l] тип,

де **прапори**:

– (мінус) - вирівнювання по лівому краю поля;

+ (плюс) - виводиться знак числа + або –;

ширина: визначає мінімальне число виведених символів, що доповнюються пробілами або нулями.

точність: визначає число знаків після десяткової точки для чисел із плаваючою точкою. Для змінних цілого типу визначається максимальне число виведених цифр.

тип:

d або **i** -десяткове ціле зі знаком (int);

u – десяткове ціле без знака (unsigned int);

x – шістнадцятирічне ціле без знака (unsigned int);

f – виведення числа з плаваючою точкою (float, double);

e – виведення числа з плаваючою точкою із використанням експоненціальної форми (float, double);

c – виведення символу (char);
s – виведення рядка;
p – виведення по вказівці.

L | l – застосовується в комбінації із символом типу:
Lf і **Le** - довгий із плаваючою точкою (long double);
ld або **li** - довгий цілий (long int);
lu – беззнаковий довгий цілий (unsigned long int).

Приклад:

```
#include <iostream>
using namespace std;

void main()
{
int g = 22;
float f = 20000/3.0f;
double d=10000.0;
long ln = 200001L;
long double ld=1.0;
d = d/3;
ld/=3;
printf(" \n-----%10.1f----",f);
//----- 6666.7----
printf(" \n-----%10.4d----",g);
//----- 0022----
printf(" \n--d=%10.5f g=%d",d,g);
//--d=3333.33333 g=22
printf(" \n--ln=%8ld ",ln);
//--ln= 200001
printf(" \n--ln=%8.7ld ",ln);
//--ln= 0200001
printf(" \n--ld=%16.15Lf ",ld);
//--ld=0.333333333333333
}
```

4.3. Об’єкт введення даних cin

Введення даних здійснюється з використанням об’єкта **cin**. Синтаксис:
cin >> змінна [>> змінна]…

При введенні рядків введення здійснюється тільки по одному слову. При введенні декількох змінних роздільник між ними - пробіл або <Enter>.

Приклад 1:

```
char f[20];
cout << "Введіть ім’я?";
```

```
cin >> f;
cout << "Ім'я:"<< f <<"\n";
int e;
cin >>e;
cout << "\ne = "<< e;
```

Приклад 2

```
#include <iostream>
using namespace std;

void main()
{
  char name[20],fam[20];
  int age;
  cin >> name >> fam >> age; //Іван Іванов 25
  cout << fam << ' ' << name << ' ' << age;
}
```

Результат роботи:

```
Іванов Іван 25
```

4.4. Функція введення даних scanf

Функція **scanf** є одною із багатьох функцій введення, які мають місце в зовнішніх бібліотеках. Кожній змінній, що вводиться у рядку функції **scanf** повинна відповідати специфікація (див. функцію виведення даних **printf**). Перед іменами змінних треба ставити символ &. Цей символ означає «взяти адресу». Більш детально про адреси та вказівки буде розглядатись у темі 8 "Вказівки, посилання та масиви".

Нижче приведено простий приклад використання функції **scanf**.

```
#include <iostream>
using namespace std;

void main()
{
 //w - вес, h - рост.
 int w,h;

 printf("Введите ваш вес: ");
 scanf("%d",&w);
 printf("Введите ваш рост: ");
 scanf("%d",&h);

 printf("\n\nВес = %d, рост = %d\n",w,h);
}
```

При введенні ваги 80 та росту 185 на екран буде виведено наступний результат:

Вес = 80, рост = 190

Контрольні питання

1. Які об’єкти введення/виведення даних вам відомі?
2. Які користувацькі маніпулятори введення/виведення ви знаєте?
3. Розкрийте призначення маніпулятору setw.
4. Розкрийте призначення маніпулятору setprecision.
5. Розкрийте призначення маніпулятору setfill.
6. Перелічить аргументи маніпуляторів setiosflags, resetiosflags.
7. Які функції введення/виведення ви знаєте?
8. Що таке специфікація формату введення/виведення? Які існують символи специфікації?
9. Наведіть приклади введення/виведення даних за допомогою об’єктів.
10. Наведіть приклади введення/виведення даних за допомогою функцій.

РОЗДІЛ 5. ОПЕРАЦІЇ В С++

При роботі з числовими даними застосовують наступні види операцій: арифметичні, операції відношення, логічні, порозрядні та додаткові операції. Дані операції працюють з арифметичними виразами, виразами відношення та логічними виразами.

Арифметичний вираз – це послідовність констант, змінних, функцій, з'єднаних знаками арифметичних операцій та скобок. Результатом арифметичного виразу є число. Вираз відношення – сукупність арифметичних виразів, з'єднаних операціями відношення. Логічний вираз – сукупність арифметичних виразів та (або) виразів відношення, з'єднаних між собою логічними операціями. Результатом виразу відношення або логічного виразу є число 1 (істина) або число 0 (неправда).

Операції виконуються відповідно до пріоритетів. Для зміни порядку виконання операцій використовуються круглі дужки. Якщо в одному виразі записано декілька операцій однакового пріоритету, унарні операції, умовна операція і операції привласнення виконуються справа наліво, останні – зліва направо. Наприклад, a = b = c означає a = (b = c), а a + b + c означає (a + b) + c.

Порядок обчислення підвиразів всередині виразів не визначений: наприклад не можна вважати, що у виразі (sin(x + 2)+ cos(y + 1)) звернення до синуса буде виконано раніше, ніж до косинуса, і що x + 2 буде обчислено раніше, ніж y+1.

Результат обчислення виразу характеризується значенням і типом. Наприклад, якщо a і b – змінні цілого типу і описані так:

```
int a = 2, b = 5;
```

тоді вираз a + b має значення 7 і тип int, а вираз a = b має значення рівне поміщеному в змінну a (у даному випадку 5) і тип, співпадаючий з типом цієї змінної. Таким чином, в С++ у виразі виду a = b = c спочатку обчислюється вираз b = c, а потім його результат стає правим операндом для операції привласнення змінної a.

У вираз можуть входити операнди різних типів. Якщо операнди мають однаковий тип, то результат операції матиме той же тип. Якщо операнди різного типу, перед обчисленнями виконуються перетворення типів по визначеним правилам, що забезпечують перетворення коротших типів у довші для збереження значущості і точності.

Перетворення бувають двох типів:

– що змінюють внутрішнє представлення величин (з втратою точності або без втрати точності);

– що змінюють тільки інтерпретацію внутрішнього представлення.

До першого типу відноситься, наприклад, перетворення цілого числа у число з плаваючою точкою (без втрати точності) і навпаки (можливо, з втратою точності), до другого – перетворення знакового цілого в беззнакове.

У будь-якому випадку величини типів char, signed char, unsigned char, short int та unsigned short int перетворяться в тип int, якщо він може представити всі значення, або в unsigned int у іншому випадку.

Після цього операнди перетворяться до найбільш довгого типу з них, і він використовується як тип результату.

Розглянемо кожен вид операцій більш детально.

5.1. Арифметичні операції в С++

Арифметичні операції: *, /, +, –, % (ділення по модулю або залишок від ділення)

```
cout << 10%2 << "\n"; //0
cout << 300%165 << "\n";   //135
```

Оператор присвоювання:

```
a = b = c = d = e = 100;
val = 3 + (r = 9 - c);
```

Складові операції:

```
+=    *=   /=   - =
```

Приклад:

```
int a = 10;
a+=10;       //20;
a-= 10;      //10;
a*=2;        //20;
a/=5;        //4;
```

Змішані типи даних:

Якщо у виразах є змішані типи даних, то виконується перетворення меншої точності у більшу. Наприклад, тип цілий і подвоєна точність забезпечує перетворення цілого типу в подвоєну точність.

```
int a = 4;
double c = 17.1567;
double t;
t = a + c;   //a-перетвориться в double
```

У ряді випадків зручно використовувати приведення типу.

Синтаксис:

(data type) expression, де
data type - будь-який тип даного
expression - змінна, константа або вираз

Приклад:

```
int c = 10;
cout << (float)c/3 <<"\n";      //3. 3333
cout << (float)(c/3);           //3
```

5.2. Операції відношення

Операції відношення (< (менше), <= (менше або дорівнює), > (більше), >= (більше або дорівнює), == (дорівнює), != (не дорівнює)) порівнюють перший операнд з другим. Операнди можуть бути арифметичного типу або вказівками. Результатом операції є значення true (1 – істина) або false (0 – неправда) (будь-яке значення, не рівне нулю, інтерпретується як true). Операції порівняння на рівність і нерівність мають менший пріоритет, ніж решта операцій порівняння.

```
cout << (2 == 2 == 2);        // 0 - неправда
cout << (2 < 3);              // 1 - істина
```

Розглянемо використання оператору if.

Оператор if

Синтаксис:

if (<вираз>)
{ блок з одного або більш операторів }

Якщо після **if** стоїть 1 оператор, фігурні дужки можна не ставити.

Оператор if ...else

Синтаксис:

if (<вираз>)
{ блок з одного або більш операторів }
else
{ блок з одного або більш операторів C++ }

$$Y = \begin{cases} X^2, & якщо \;\; X > 0 \\ 2X, & в\ інших\ випадках \end{cases}$$

```
if (X > 0) Y = X * X;
else Y = 2 * X;
```

5.3. Логічні операції

Логічні операції:

&& – **І**
|| – **АБО**
! – **НЕ**

Операнди логічних операцій **І** (&&) та **АБО** (||) можуть мати арифметичний тип або бути вказівками, при цьому операнди в кожній операції можуть бути різних типів. Перетворення типів не виконується, кожен операнд оцінюється з точки зору його еквівалентності нулю (операнд, рівний нулю, розглядається як false, не рівний нулю, – як true).

Результатом логічної операції є true (1) або false (0). Результат операції логічне **І** має значення true тільки тоді якщо обидва операнди мають значення true. Результат операції логічне **АБО** має значення true, якщо хоча б один з операндів має значення true. Логічні операції виконуються зліва направо. Якщо значення першого операнда досить, щоб визначити результат операції, другий операнд не обчислюється. Такий засіб називається усіканням.

if (a > b || c > d || e > f) – переривається, як тільки обчислюється, що **a > b**, або **c > d**.

Логічне заперечення (!) дає в результаті значення 0 (false), якщо операнд є істиною (не нуль), та значення 1 (true), якщо операнд рівний нулю.

!(var1 = = var2) – те ж саме **var1 ! = var2**

Пріоритетність операцій:

- арифметичні операції;
- операції відношення;
- логічні операції.

Усікання може бути використане для коректного обчислення.

Наприклад:

if (b ! = 0 && a / b > 12.4) - коректно, ділення на нуль не буде.

Якщо записати **if (a / b > 12.4 && b ! = 0)** – небезпечно, може бути ділення на нуль.

Логічні вирази повертають значення типу bool. Будь-яке ненульове значення розглядається як істина.

5.4. Додаткові операції

Умовна операція ?

Синтаксис:

результат = вираз1 ? вираз2 : вираз3

Перший вираз може мати арифметичний тип або бути вказівкою. Він оцінюється з погляду його еквівалентності нулю (вираз, рівний нулю, розглядається як false, не рівний нулю, – як true). Якщо результат обчислення виразу 1 рівний true, то результатом умовної операції буде значення другого виразу, інакше – третього виразу. Обчислюється завжди або другий вираз або третій. Їх тип може розрізнятися. Умовна операція є скороченою формою умовного оператора if … else.

Приклад:

```
#include <iostream>
using namespace std;
```

```
void main()
{
 int a = 6, b = 8;
 int min = ( a < b ) ? a : b;
 cout << "min = " << min; // 6
}
```

Операція інкремент (++) і декремент (--)

Ця операція збільшує або зменшує значення змінної.

При цьому в залежності від розташування справа або зліва від змінної інкремент **++** і **--** називають префіксним або постфіксним.

`++i` означає `i = i + 1` або `i += 1;`
`--i` означає `i = i - 1` або `i -= 1;`

Якщо змінна має префіксну операцію, вона змінює своє значення перед тим, як буде використана у виразі. Якщо використовується постфіксна операція, то змінна змінить своє значення після обробки виразу:

```
a = 6;
b = a++ -1;   // a = 7  b = 5
```

Префіксна операція має вищий пріоритет, ніж будь-яка інша операція.

`s = ++(a + b );` → **Помилка!!!**

Операція застосовується тільки до змінної.

Приклад:

```
#include <iostream>
using namespace std;

void main()
{
int i = 1, j = 2, k = 3, a;
a = i++ + j - --k;
cout << a << i << k;     // 122
}
```

Операція sizeof()

Операція визначення розміру **sizeof** призначена для обчислення розміру об’єкту або типу даних у байтах та має дві форми:

sizeof вираз
sizeof (тип даних)

Приклад:

```
#include <iostream>
using namespace std;

void main()
{
 float x = 3;
 cout << "sizeof(float) = " << sizeof(float);
 cout << "\nsizeof(x) = " << sizeof x;
 cout << "\nsizeof(x + 5.0) = " << sizeof(x + 5.0);
}
```

Результат роботи програми:

```
sizeof(float) = 4
sizeof(x) = 4
sizeof(x + 5.0) = 8
```

5.5. Порозрядні операції

Таблиця 5.1

Основні порозрядні операції

Операція	Значення
&	Порозрядне І
|	Порозрядне АБО
^	Порозрядне виключаюче АБО
~	Порозрядне заперечення (доповнення до 1)

Таблиця 5.2

Характеристика основних порозрядних операцій

1-й біт	Операція	2-й біт	Результат
1	&	1	1
1	&	0	0
0	&	1	0
0	&	0	0
1	|	1	1
1	|	0	1
0	|	1	1
0	|	0	0
1	^	1	0
1	^	0	1
0	^	1	1
0	^	0	0
	~	1	0
	~	0	1

```
  9 & 14     // 8
 1 0 0
1
 1 1 1
0
-------
 1 0 0
0
  9 | 14     // 15
1  0  0
1
 1 1 1
0
-------
 1 1 1
1

  9 ^ 14      // 7
 1 0 0
1
 1 1 1
0
-------
 0 1 1
1

  ~ 9         // 6
 1 0 0
1
-------
 0 1 1
0
```

Приклад роботи з оператором порозрядного заперечення ~.

```
#include <iostream>
using namespace std;

void main()
{
 unsigned char u1 = ~9;     // 246
 signed char u2 = ~9;       // -10
 unsigned int u3 = ~9;      // 65526
 signed int u4 = ~9;        // -10
 unsigned long u5 = ~9;     // 4294967286
 signed long u6 = ~9;       // -10

 cout<<"u1 = "<<(int)u1<<"\n";
 cout<<"u2 = "<<(int)u2<<"\n";
```

```
 cout<<"u3 = "<<u3<<"\n";
 cout<<"u4 = "<<u4<<"\n";
 cout<<"u5 = "<<u5<<"\n";
 cout<<"u6 = "<<u6<<"\n";
}
```

Приклад визначення парного або непарного значення

```
#include <iostream>
using namespace std;


void main()
{
 int input;
 cin>> input;
 if(input & 1) // Істина, якщо число непарне
   cout<< "Число непарне";
 else
   cout<< "Число парне";
}
```

У ASCII кодах єдина відзнака між маленькими і великими буквами значення 0 або 1 у 5-му біті - для англійських букв.

Біт → 7 6 5 4 3 2 1 0
A → 0 1 0 0 0 0 0 1 → Шест. 41, Дес. 65
a → 0 1 1 0 0 0 0 1 → Шест. 61, Дес. 97

Щоб перетворити символ до верхнього регістру необхідно встановити 0 у 5-й біт.

Для цієї мети можна використовувати маску зі значенням

$32_{(10)} = 20_{(16)} = 1\ 0\ 0\ 0\ 0\ 0_{(2)}$ → бітова маска.

Наведемо приклад переключення між маленькими та великими буквами, використовуючи порозрядні операції.

Приклад:

```
#include <iostream>
using namespace std;

#define bitmas (0x20) // 100000

void main()
{
 char a,b,c;
 cin>>a; //g
```

```
    cin>>b; //M
    cin>>c; //p

    //Виключення 5-го біту (0) (великі букви)
    a = a & ~bitmas;
    b &= ~bitmas;     // Складові порозрядні
    c &= ~bitmas;     // операції
    cout<<a<<b<<c<<"\n";  //GMP

    //Включення 5-го біту (1) (маленькі букви)
    a = a | bitmas;
    b |= bitmas;     // Складові порозрядні
    c |= bitmas;     // операції
    cout<<a<<b<<c<<"\n";  //gmp

    //Переключення 5-го біту
    a = a ^ bitmas;
    b ^= bitmas;     // Складові порозрядні
    c ^= bitmas;     // операції
    cout<<a<<b<<c<<"\n";  //gmp
}
```

Для переключення 5-го біта з 1 у 0 (виключення 5-го біту) використовується порозрядний оператор **І** (“&”) разом з оператором порозрядного заперечення (“~”). У випадку переключення 5-го біту з 0 у 1 (включення 5-го біту) використовується порозрядний оператор **АБО** (“|”). Для переключення заданого біта з 0 у 1 і навпаки, ефективно використовувати виключаюче **АБО** (“^”).

У даному прикладі спочатку виключається 5-й біт (перетворюється у 0) та в результаті отримаємо три маленькі букви “gmp”. Потім 5-й біт включається (перетворюється у 1) – отримуємо три великі букви “GMP”. На останньому етапі переключаємо отриманий результат, використовуючи виключаюче **АБО** (“^”) (1 перетворюється у 0) та отримуємо знову три маленькі букви “gmp”.

У операційній системі Windows використовуються порозрядні операції для додавання або виключення атрибутів вікна.

Оператор “виключаючи **АБО** (^)” використовується у двовимірній графіці GDI при роботі з піксельними операціями, а також в сучасних криптографічних алгоритмах.

Порозрядні операції зсуву.

Синтаксис:

a << b → зсув ліворуч

a >> b → зсув праворуч

Приклад:

```
29 << 3                  // 232
0 0 0 1 1 1 0 1          // 29
1 1 1 0 1 0 0 0          // 232
```

← зсув на 3

Ця операція дорівнює $29 \cdot 2^3 = 232$

Початковий знак від'ємного числа зберігається. Порозрядний зсув для від'ємного числа робиться наступним чином. При зсуві числа ліворуч "<<" від'ємний знак числа зберігається, а в кінці додаються нулі. При зсуві від'ємного числа праворуч старші розряди заміняються одиницею, а не нулем.

Розпишемо арифметику порозрядного зсуву для додатних та від'ємних чисел.

Якщо **x** додатне число, тоді

$x << n = x * 2^n$
$x >> n = x / 2^n$ (цілочисельне ділення)

Якщо **x** від'ємне число, тоді

$x << n = -(|x| * 2^n)$
$x >> n = -(|x| / 2^n) - 1$ ($|x| / 2^n$ – цілочисельне ділення)

Приклад порозрядних зсувів.

```
#include <iostream>
using namespace std;

void main()
{
 int num_1 =  25;          // 0000000000011001
 int num_2 = -25;          // 1111111111100111
 int shift_1, shift_2, shift_3, shift_4;
 shift_1 = num_1 << 3;     // 0000000011001000     200
 shift_2 = num_1 >> 3;     // 0000000000000011       3
 shift_3 = num_2 << 3;     // 1111111100111000    -200
 shift_4 = num_2 >> 3;     // 1111111111111100      -4
 cout << shift_1 << "\t" << shift_2 << "\n";
 cout << shift_3 << "\t" << shift_4 << "\n";
 //  200    3
 // -200   -4
}
```

Може бути використана складова операція порозрядного зсуву:

<<= – складений порозрядний зсув ліворуч

>>= – складений порозрядний зсув праворуч

```
signed int num1 = 15;
unsigned int num2 = 15;
num1 <<= 4; //множення на 16 (240)
num2 >>= 3; //ділення на 8 (1)
```

Контрольні питання

1. Які ви знаєте арифметичні операції?
2. Призначення арифметичної операції %.
3. Наведіть приклади складних арифметичних операцій.
4. Які ви знаєте операції відношення?
5. Які ви знаєте логічні операції?
6. Чим відрізняються логічні операції від порозрядних.
7. Опишіть додаткову операцію “?”.
8. Наведіть приклади порозрядних операцій зсуву “<<”, “>>”.
9. Розкрийте сутність операцій інкременту та декременту. Наведіть приклади.
10. Чим відрізняються префіксні та постфіксні операції?

Завдання

Лінійний обчислювальний процес

Вибравши варіант індивідуального завдання, необхідно:

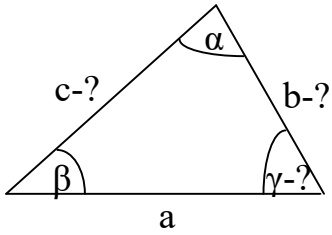
1. Скласти структурну схему алгоритму.
2. Скласти програму.
3. Підібрати контрольний приклад, що забезпечить перевірку правильності складання програми.

При виконанні завдання необхідно використовувати графічний засіб запису алгоритму, що представляє собою процес вирішення задачі у вигляді структурної схеми. Послідовність дій при виконанні алгоритму являє собою набір фігур-блоків, розташованих зверху вниз та з’єднаних лініями. Виконана дія записується всередині блоку визначеної форми. Блоки мають наскрізну нумерацію. Програмування здійснюється на мові С++. Бажано, щоб програма була орієнтована на діалоговий режим роботи. Для цього, перед тим як запросити у програмі введення даних з клавіатури, необхідно вивести повідомлення-підказку, наприклад "Введіть значення А". Виведення результатів необхідно супроводжувати пояснювальним текстом.

Приклади виконання блок-схем у лінійному, розгалуженому та циклічному процесах представлені у темі 13 "Основи алгоритмізації обчислювальних процесів" даного посібника.

Приклад виконання завдання з лінійного обчислювального процесу.

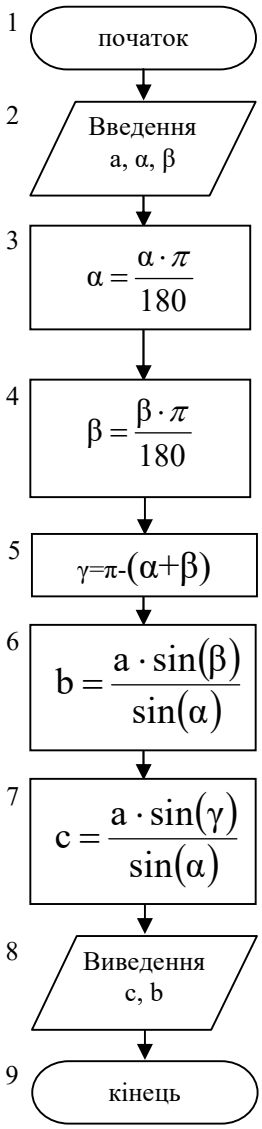
Знайти дві сторони трикутника b і c по відомій довжині сторони a і двом внутрішнім кутам α і β.



Код програми має наступний вигляд:

```
#include <iostream>
#include <cmath>
using namespace std;

#define M_PI        3.14159265358979323846

void main()
{
 double alpha,beta,gamma;
 double a,b,c;

 cout<<"Введіть сторону a ";
 cin>>a;
 cout<<"Введіть кут alpha ";
 cin>>alpha;

 cout<<"Введіть кут beta ";
 cin>>beta;

 alpha = alpha*M_PI/180;
 beta = beta*M_PI/180;
 gamma = M_PI - (alpha + beta);
 b = a*sin(beta)/sin(alpha);
 c = a*sin(gamma)/sin(alpha);

 cout<<"\nb = "<<b;
 cout<<"\nc = "<<c;

}
```

Наведемо блок-схему вирішення даної задачі.



Варіанти індивідуальних завдань

1. Визначити значення функції y

1. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \sqrt{\left|1.2A + \ln\left|x\right| + 1\right|} \cdot e^{x}$.
2. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = 0.1B^{2} + B \cdot \sin x + 2.3 \cdot e^{\cos x}$.
3. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = 0.5A - \left|x\right|^{\sin\frac{x}{2}} + \ln\left|x\right| + 1\Big|$.
4. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \frac{A}{2x^{2} + x + 0.6}\sin x + e^{A}$.
5. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \frac{B}{0.5\left|x\right| + 3} + \ln\left|B\right| + 1\Big| \cdot \sin x^{2} + 15.2$.

6. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \left(\left(\frac{B}{|x|+1} \right)^{\frac{1}{|B|+3}} - 1 \right) A \sin x + e^{\cos B}$.

7. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \frac{0.5}{|A \sin x(x+1.1)| + 2.5} \cos x - \ln||A|+1|$.

8. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \frac{Ae^x}{|A(\sin x + arctg\ A)|+1} + \ln||x|+1|$.

9. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \frac{2}{B\sqrt{B^2+C^2}+1} \cdot \left(\frac{(B^2+C^2) \cdot \sin(\frac{Bx}{2})}{\sqrt{B^2+C^2}+1} \right)$.

10. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = -\frac{2}{|\pi \cdot A|+2} \cdot \ln \left| \frac{A+\sqrt{\pi^2+x^2}}{\left|A-\sqrt{\pi^2-x^2}\right|+1} \right|$.

11. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \frac{\sin 2x + \cos x}{\lg(x^2+2)} e^x$.

12. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \frac{\pi + (Ax+B)^2}{\left|A-(\pi x+B)^2\right|+1} \cdot \frac{\sqrt[5]{(\pi x+B)^2}}{\left|\sqrt[3]{(Ax+B)^2}\right|+1}$.

13. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \frac{1+\cos(x-2)}{2+\frac{x^4}{2}+\sin^2 x}$.

14. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = (\sin^2(\pi x)-2)^{2x} - tg(\sin^2(\pi x)-2)^{4x}$.

15. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \sqrt[3]{2 \cdot \ln(\pi - \sin^2 x)^2} - \sqrt[3]{\lg(\pi - \sin^2 x)^4}$.

16. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = Ae^{2x+1} \cdot \frac{6.3 \cdot \cos x}{\sin^2 x + 1}$.

17. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \lg \left| 4 - \cos^2 \frac{x}{4} \right| - 2 \ln \left(4 - \cos^2 \frac{x}{4} \right)^2$.

18. Скласти програму та блок-схему алгоритму лінійного обчислювального

виразу: $y = \dfrac{e^{2x} + e^{0.57x}}{\lg\left(A^2 + 2\right)} + \sin^2 x$.

19. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \ln x - \dfrac{x - 100}{0.13 \cdot \sin x} + 8.2 \cdot e^x$.

20. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \dfrac{A \cdot \sin x}{1 + \left|\lg(2x^2 + 2)\right|} + e^x$.

21. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \lg\left|\dfrac{Ax}{(A - x)^2 + 1} + 1\right| + e^{2x-1}$.

22. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = x\left|\ln\left(x^2 + 2\right) + \dfrac{1}{100}\right|^A - \sin^2 x$.

23. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \dfrac{Ae^x}{\left|A(\sin x + arctgx)\right| + 1} + \ln\left(x^2 + 3\right)$.

24. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \dfrac{\left|A(x + 1)\right|^{\frac{\pi}{2}}}{|2x| + 3} + \ln^2\left(x^2 + 5\right)$.

25. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \dfrac{2x}{\left|A \cdot \sin x \cdot (x + A)\right| + 1}\cos x - \ln\left(A^2 + 4\right)$.

26. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \dfrac{\sqrt[3]{\left|x^2 \cos^2 x\right|}}{13.2 \cdot \lg(x^2 + 1.1)}$.

27. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \dfrac{\lg(x^2 + x^4 + 1)}{\sqrt{2x^2 + \cos^2 x}}$.

28. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \dfrac{\sqrt{\left|\lg(x^4 + A^2)\right|}}{\left|Ae^x \sin x\right| + 1}$.

29. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = \dfrac{\sin^4 x + \cos^2 x - \ln\left(x^2 + 3\right)}{e^x}$.

30. Скласти програму та блок-схему алгоритму лінійного обчислювального виразу: $y = (1 + x) \cdot \dfrac{x + \dfrac{x}{x^2 + 4}}{e^{-x-2} + \dfrac{1}{x^2 + 4}}$.

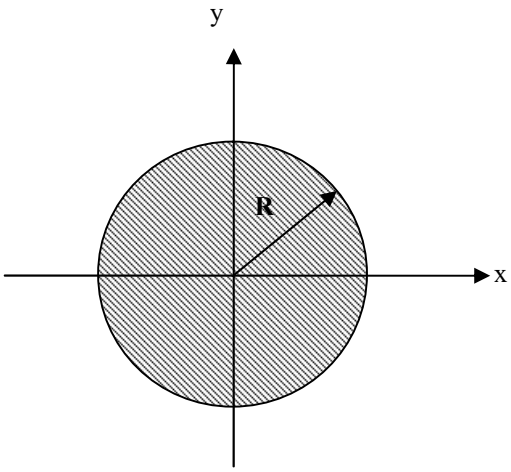
2. Геометричні завдання

1. Обчислити площу кільця, утвореного двома концентричними колами відомих радіусів.
2. Обчислити площу трикутника за трьома заданими сторонами.
3. Обчислити площу трикутника за заданими двовимірними координатами усіх його вершин.
4. Задана діагональ квадрата. Обчислити його площу.
5. На площині задані точка $A^{(x_a, y_a)}$ та коло радіусу R з центром в точці $B^{(x_b, y_b)}$.Обчислити відстань від точки А до найближчої точки кола.
6. За відомою довжиною ребер прямокутного паралелепіпеду обчислити площу його поверхні та об'єм.
7. За відомою площею квадрата, який уписаний в круг, обчислити площу круга.
8. Обчислити відстань між двома точками з відомими трьохвимірними координатами.
9. За відомою величиною діагоналі грані кубу обчислити значення його об'єму.
10. За відомою величиною площі круга, який уписаний в квадрат, обчислити значення площі квадрату.
11. За відомою величиною об'єму кулі, яка уписана в куб, обчислити значення об'єму куба.
12. За відомою довжиною ребра куба обчислити довжину внутрішньої діагоналі.
13. За відомою довжиною кола обчислити площу уписаного правильного шестикутника.
14. За відомим периметром правильного уписаного шестикутника обчислити площу круга.
15. За відомою площею правильного шестикутника обчислити площу уписаного круга.
16. Обчислити суму цифр заданого натурального трьохзначного числа.
17. Обчислити добуток цифр заданого натурального трьохзначного числа.
18. Обчислити залишок від ділення натурального двозначного числа на суму цифр цього числа.

Розгалужений обчислювальний процес

Приклад виконання завдання з розгалуженого обчислювального процесу.

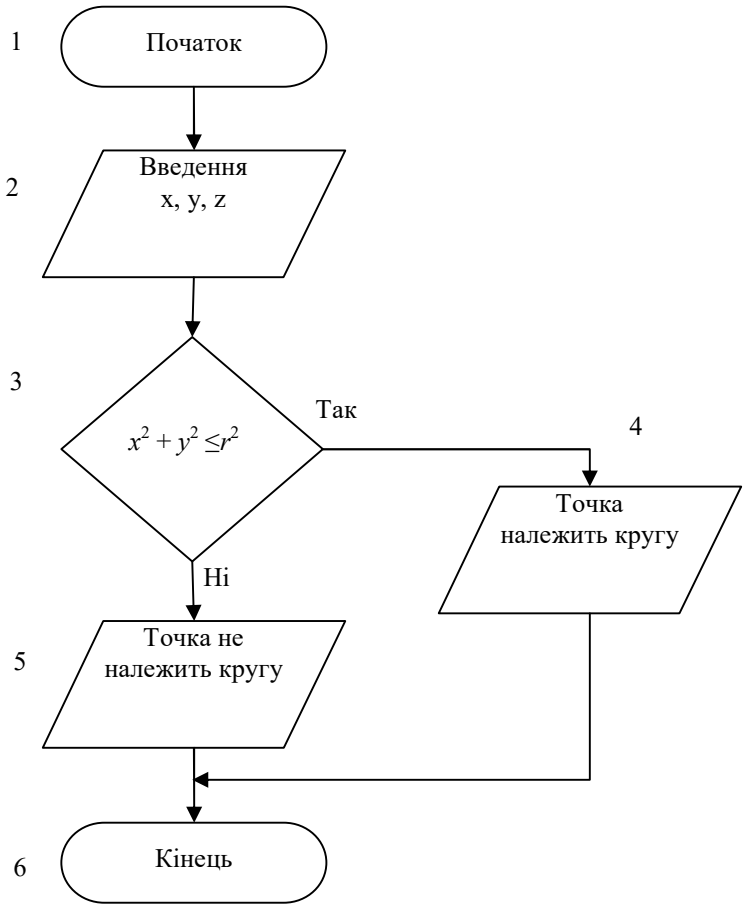
Перевірити, чи міститься точка з заданими координатами усередині заштрихованої області.



Код програми має наступний вигляд:

```
#include <iostream>
using namespace std;

void main()
{
 double x,y,r;

 cout<<"Введіть радіус круга ";
 cin>>r;
 cout<<"Введіть координати точки:\nX ";
 cin>>x;
 cout<<"Y ";
 cin>>y;

 if (x*x + y*y <= r*r)
    cout<<"\nТочка належить кругу";
 else
    cout<<"\nТочка не належить кругу";
}
```

Наведемо блок-схему вирішення даної задачі.



Варіанти індивідуальних завдань

1. Визначити значення функції y

1. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y = \begin{cases} x\sqrt{B^3} & при \quad 0 < x < 4 \quad та \quad B > 0 \\ B\sin^2 x & при \quad x \ge 4 \quad та \quad B = 1 \\ 0 & в \quad інших \quad випадках \end{cases}$$

2. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y = \begin{cases} tgx & при \quad 0 < x \le \dfrac{\pi}{2} \\ Bctgx & при \quad \dfrac{\pi}{2} < x < \pi \\ 5 & в \quad інших \quad випадках \end{cases}$$

3. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y = \begin{cases} x^2 - 5 & при \quad x > 2 \\ \lg x + 4 & при \quad x = 2 \\ x^3 - 2 & в \quad інших \quad випадках \end{cases}$$

4. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}2^{-x\sqrt{|x-2|}} & при \quad x<0\\ \sqrt{e^{x+1}+\sin z} & при \quad 0<x<5 \quad та \quad z=2\\ 0 & в \quad інших \quad випадках\end{cases}$$

5. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}2^{x}+3z & при \quad x<0,\quad z>0\\ e^{|x-z|}+x/2 & при \quad x<0,\quad z<0\\ x^{2} & в \quad інших \quad випадках\end{cases}$$

6. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\dfrac{x-z/2}{2(x+z)} & при \quad z\le 0 \quad та \quad x\ne 0\\ x/\sin z & при \quad 1>z>2 \quad та \quad x>0\\ x^{z} & в \quad інших \quad випадках\end{cases}$$

7. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}x^{3}(\ln x+4) & при \quad x>0\\ 0 & при \quad x=0\\ x^{2} & в \quad інших \quad випадках\end{cases}$$

8. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\sin x-\cos x & при \quad -1\le x<1\\ \sqrt[3]{x^{2}}+2x-\ln|x| & при \quad x<-3\\ \sin x & в \quad інших \quad випадках\end{cases}$$

9. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}x^{z+1}+e^{x-1} & при \quad x>0\\ \dfrac{|z-x|^{3}}{2+\operatorname{tg} z} & при \quad x<0 \quad и \quad 1<z<2\\ xz & в \quad інших \quad випадках\end{cases}$$

10. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\ln^{2}(x+2) & при \quad x>0\\ x/2+\sin^{3} z & при \quad x\le 0,\ 1<z<2\\ \cos x & в \quad остальных\ случаях\end{cases}$$

11. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}(x+z)/(5+\sin^2 z) & при \quad x>0\\ \sqrt{|x+\operatorname{tg}^2 z|} & при \quad x<0,\quad 0<z<3\\ z\cdot e^x & в \quad інших \quad випадках\end{cases}$$

12. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}e^{-(x+3)}+\operatorname{arctg} z & при \quad x>0,\quad -\pi/2\le z\le \pi/2\\ 2x/\lg(x^2+3) & при \quad x<0,\quad z>\pi/2\\ \sqrt{|x|} & в \ інших \ випадках\end{cases}$$

13. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\sqrt[3]{|x|+|z|}/\ln(x+1) & при \quad x>0\\ \left|(x^3-2z)/(x+z)^2\right| & при \quad x<0,\quad z<0\\ tgx & в \ інших \ випадках\end{cases}$$

14. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\sqrt{|x|+|z|} & при \quad x>0\\ x^3\cdot\ln z & при \quad x<0,\quad z>0\\ e^{|x|} & в \ інших \ випадках\end{cases}$$

15. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\sqrt[3]{|x-z|} & при \quad 0\le x\le 5\\ \sin x+\cos^2 x & при \quad x>5\\ x^2z^2 & в \ інших \ випадках\end{cases}$$

16. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$P=\begin{cases}e^x+e^z & при \quad 2\le z\le 5,\quad x>0\\ 0.5\cdot\operatorname{arctg}(\ln|z|) & при \quad -\pi/2\le z\le \pi/2\\ \sin xz & в \ інших \ випадках\end{cases}$$

17. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\lg(xz)+\cos x-e^{xz} & при \quad 0<x<1.5,\quad z>0\\ \sqrt{|x^3+zx|}+e^{zx} & при \quad x\ge 1.5\\ |x|^{\sin z} & в \ інших \ випадках\end{cases}$$

18. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\dfrac{x(A+B)^{\frac{\pi}{2}}}{A}+\ln x \quad при \quad 1<x\le 3, \quad A\ne 0\\ \dfrac{\sqrt{A+B\lg x}}{B}+\sin^2 x \quad при \quad x>3, \quad A>0, \quad B>0\\ \lg|x| \quad в\ інших\ випадках\end{cases}$$

19. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\dfrac{\cos^2 x+A\lg(x^2)}{\pi}\sqrt{Ax} \quad при \quad x>0, \quad A>0\\ 0 \quad при \quad x=0, \quad A=0\\ A\sin x+A^2x \quad в\ інших\ випадках\end{cases}$$

20. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\dfrac{e^x}{A-B}+\sin^2 x \quad при \quad 0<x\le 1, \quad A\ne B\\ \dfrac{1-e^{ax}+e^{bx}}{A} \quad при \quad x<0, \quad A\ne 0\\ A\cos x+Ax^2 \quad в\ інших\ випадках\end{cases}$$

21. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}B\lg A+x^3 \quad при \quad x\le 5, \quad A>1\\ \dfrac{Ax+B^2}{x^2A} \quad при \quad x\ge 6, \quad x\ne 0, \quad A\ne 0\\ \sin x \quad в\ інших\ випадках\end{cases}$$

22. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\sqrt{B^2+C\sin x}+e^x \quad при \quad x\ge 0, \quad b>1\\ e^A+e^B+e^C \quad при \quad x=0\\ \sqrt{|Ax^2+Bx+C|} \quad в\ інших\ випадках\end{cases}$$

23. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\dfrac{\sin Bx+\cos^2 x}{|\ln(A+x)|+1}e^x \quad при \quad 0<x\le 1, \quad A>0\\ \dfrac{tgBx}{x(A-B)} \quad при \quad x\ge 2, \quad A\ne B\\ A\sin(\pi x)+e^x \quad в\ інших\ випадках\end{cases}$$

24. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\dfrac{Bx^2}{Ax-B}e^{Bx} & при \quad 0<x<2, \quad Ax\neq B\\ \left(Ax^2+e^{Bx}\right)\sin x & при \quad 2<x<3\\ \sin(Ax)^2 & в\ інших\ випадках\end{cases}$$

25. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\dfrac{B^x+A^x}{x^2}\lg Ax & при \quad 0.5<x\le 2, \quad A>0\\ \dfrac{\sqrt[3]{A^2\cos^2(Bx)}}{A^2+x} & при \quad x>2.5, \quad A^2\neq x\\ \cos Ax & в\ інших\ випадках\end{cases}$$

26. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}\dfrac{A\cos x+e^x}{Bx+A} & при \quad 0.5<x<2, \quad Bx\neq A\\ e^{Ax}+B^x+\lg x & при \quad x>2.5, \quad -1<A\le 2\\ 5x^2+AB & в\ інших\ випадках\end{cases}$$

27. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}A^x+B^{x-A}+e^{Ax} & при \quad 1<x\le\pi, \quad A>0, \quad B>0\\ A\sin^2 x+B\cos x^2+ctgx & при \quad -1<x\le 1\\ \sqrt{|Ax+B|} & в\ інших\ випадках\end{cases}$$

28. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

$$y=\begin{cases}A\lg x+B\ln x+C^x & при \quad 1<x\le 2.5, \quad C>0\\ e^{Ax+B}-AC & при \quad 2.5<x<4, \quad A>C\\ A\cos x+B\sin x-C & в\ інших\ випадках\end{cases}$$

29. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

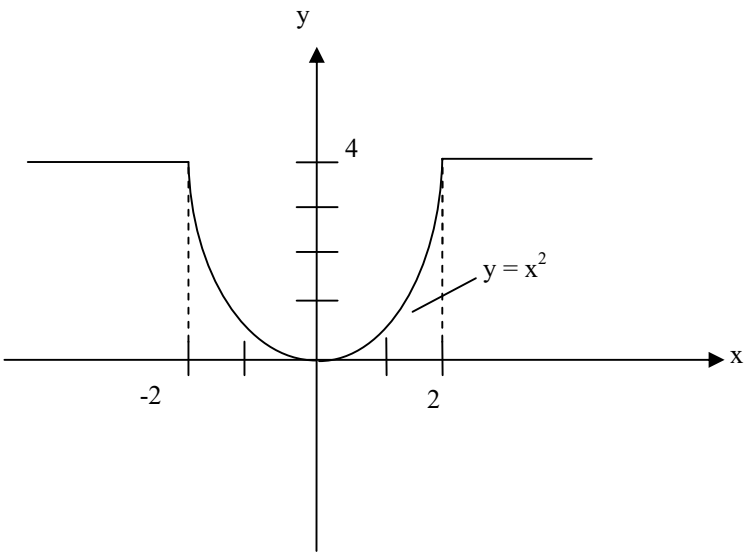
$$y=\begin{cases}C^3\lg x+A\sin x & при \quad 1.1<x<4, \quad A>0, \quad C>0\\ \dfrac{A\sin^2 x+C^3}{C^2-A} & при \quad -1<x\le 1.1, \quad C^2\neq B\\ \cos x & в\ інших\ випадках\end{cases}$$

30. Скласти програму та блок-схему алгоритму розгалуженого обчислювального виразу:

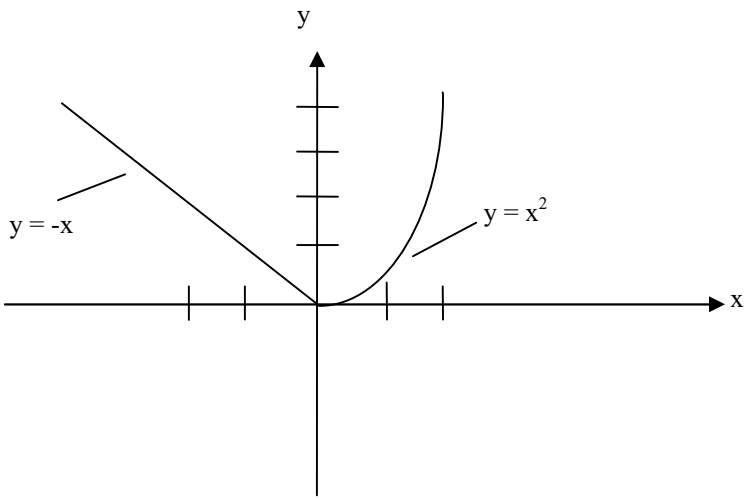
$$y=\begin{cases}e^{\lg x}+A^{b-x} & при \quad 1<x\le 2.4, \quad A>0\\ A\sin(Bx+e^{Cx}) & при \quad -1<x\le 1, \quad C>0\\ \sqrt{|Ae^x|} & в\ інших\ випадках\end{cases}$$

2. Графіки залежностей $y = f(x)$ наведені на малюнках. По заданому x визначити y.

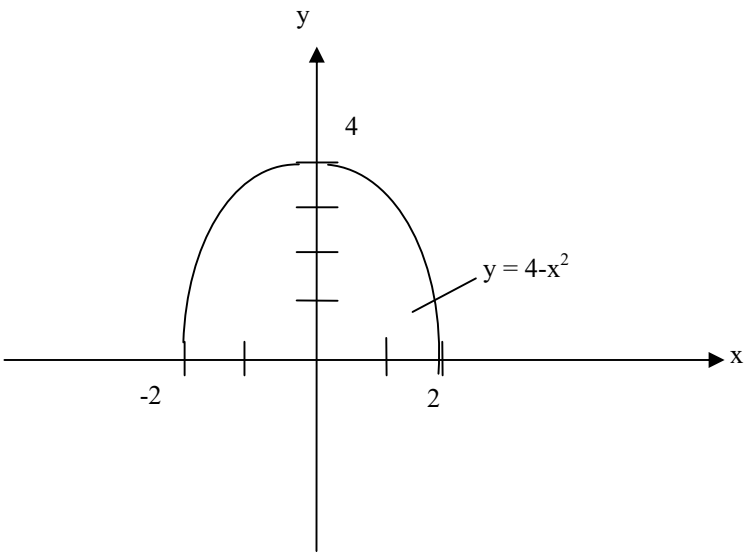
1.



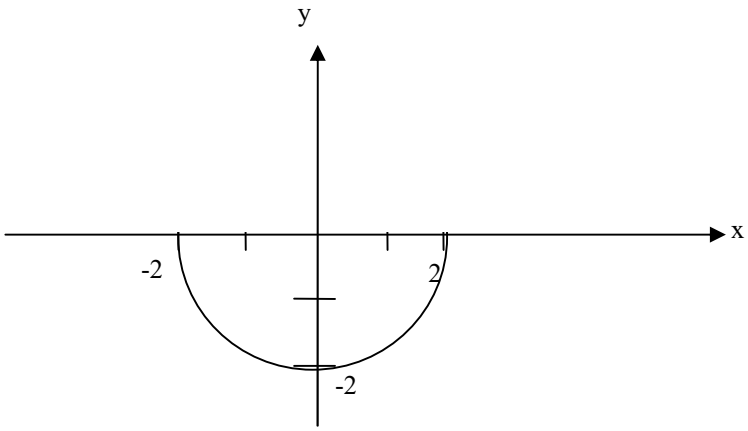
2.



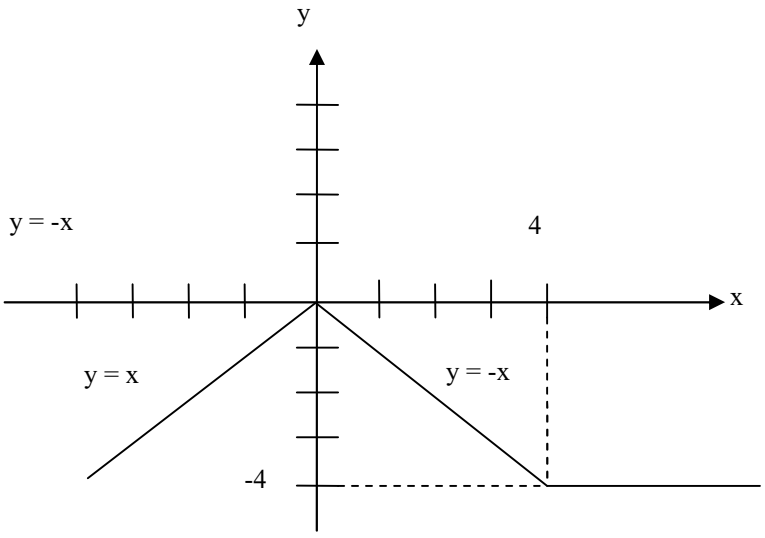
3.



4.

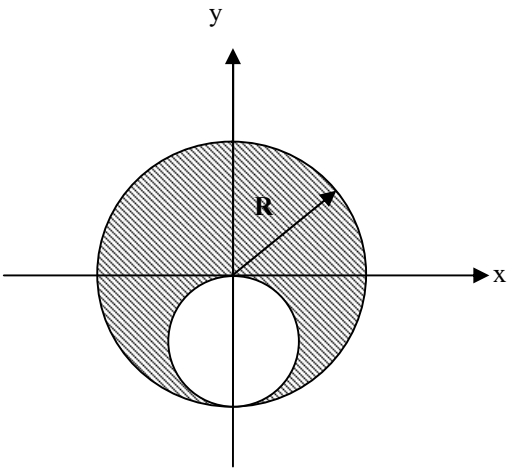


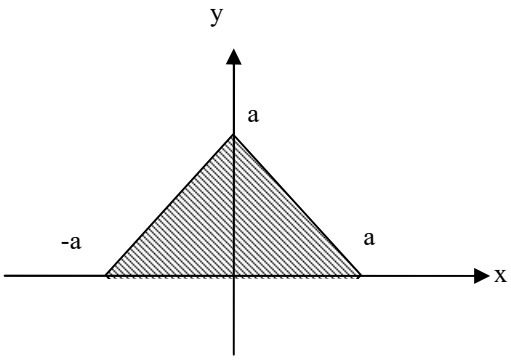
5.



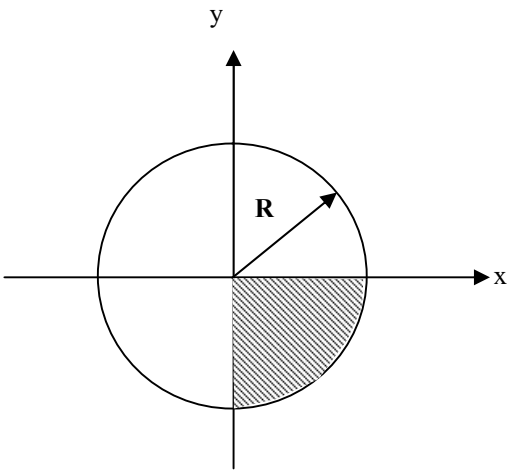
3. Перевірити, чи міститься точка з заданими координатами усередині заштрихованої області.

1.

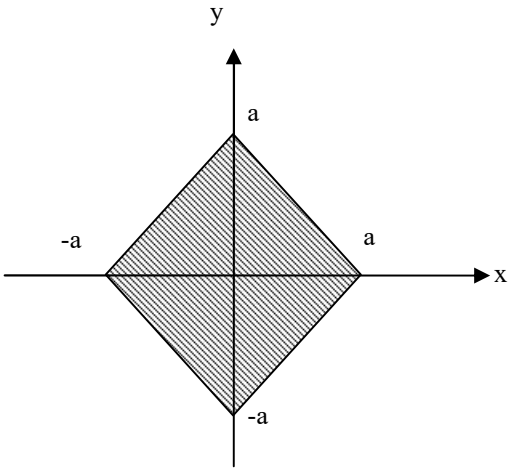




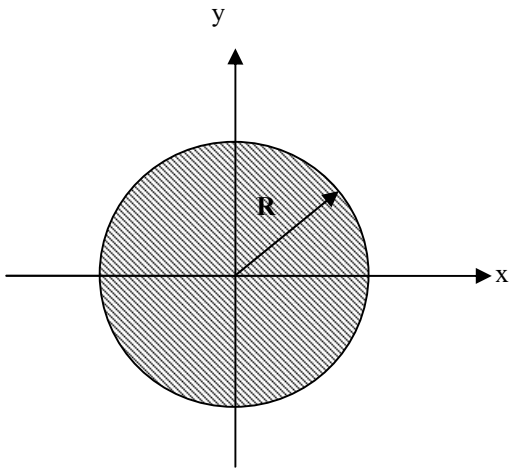
2.

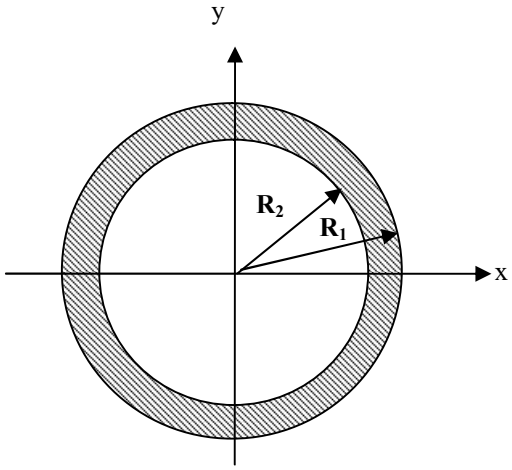


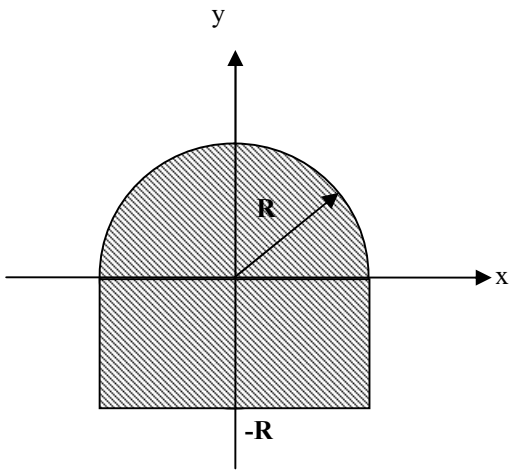
3.



4.

5. 

6. 

7. 

8. Оцінити приналежність двох заданих відрізків одній прямій.
9. Знайти точки перетину двох кіл.
10. Оцінити положення двох заданих прямокутників (не перетинаються, один в іншому, перетинаються).
11. Оцінити положення двох відрізків (не перетинаються, перетинаються, накладаються).
12. Оцінити, як задані координати трикутника (за годинниковою стрілкою або проти).

РОЗДІЛ 6. ОРГАНІЗАЦІЯ ЦИКЛІВ

Цикли використовуються для обчислень, які повторюються багато разів. Будь-який цикл складається з тіла циклу, тобто тих операторів, які виконуються декілька разів, початкових установок, модифікації параметру циклу та перевірки умови продовження виконання циклу. Цикли бувають двох типів: арифметичний (оператор **for**) та ітераційний (**while, do…while**). У арифметичному циклі заздалегідь відома кількість ітерацій, у ітераційному – невідома. Ітераційні цикли бувають з передумовою (**while**) та післяумовою (**do…while**). Різниця між ними в тому, що тіло циклу з післяумовою завжди виконується хоча б один раз, після чого перевіряється, чи треба його виконувати ще раз. Перевірка виконання циклу з передумовою робиться до тіла циклу, тому можливо, що він не виконається жодного разу.

6.1. Організація арифметичних циклів з використанням оператору for

Синтаксис арифметичного циклу **for**:

for ([початковий вираз]; [контрольний вираз]; [лічильник])
{
блок з одного або більш операторів;
}

Якщо в блоці один оператор – фігурні дужки можна не ставити.

Початковий вираз визначається один раз. Якщо **контрольний вираз** не дорівнює 0, то він є істинним і виконується блок. Цикл припиняє роботу, якщо **контрольний вираз** дорівнює 0. Вираз **лічильник** обчислюється при кожному повторенні циклу.

Приклади:

```
for(i = 10; i ! = 0; i--) або for(i = 10; i; i--)

char i;
for(i = '1'; i <= 'Z'; i++)

for(; ;) //- порожньо - вічний цикл
{ cout << "12"; }

int i = 1;
for(;i <= 100;)
{
cout << i;
i++;
}
```

```
int i;
for(i = 1; i <= 10; i++)
cout << i << '\n';
```

Результат:

```
1
2
…
9
10
```

Вкладені цикли:

```
for(i = 1; i <= 3; i++)
{
  for(j = 1; j <= 3; j++)
  {
    cout << j << '\n';
  }    //У цьому фрагменті фігурні дужки
}      //можна не ставити
```

Результат:

```
1
2
3
1
2
3
1
2
3
```

Додатково використовуються оператори:

break;
continue;

Оператор **break** завершує цикл достроково.

Приклад:

```
#include <iostream>
using namespace std;

void main()
{
 int num;
```

```
  char i;
  cout << "Виведення 20 чисел \n";
  for( num = 1; num <= 20; num++ )
  {
    cout << num << '\n';
    cout << "Будете виводити наступне число (Y/N)?";
    cin >> i;
    if((i == 'N') || (i == 'n')) break;
  }
}
```

Оператор **continue** повертає управління до початку циклу.

Приклад:

Тільки маленькі англійські літери при введенні перетворяться у великі та будуть виводитись на екран.

```
#include <iostream>
using namespace std;

void main()
{
  char m;
  int i;
  for(i = 1; i <= 5; i++)
  {
    cin >> m;
    if(( m < 'a') || ( m > 'z')) continue;
    m-= 32;
    cout << m << '\n';
  }
}
```

6.2. Організація ітераційних циклів з передумовою while та післяумовою do...while

Синтаксис циклу з передумовою **while**:

while (<вираз>)
{
блок із 1-го або більш операторів
}

while працює доти, поки вираз не буде дорівнює 0 (false). Якщо **блок** складається з 1-го оператора, фігурні дужки можна не ставити.

Приклад 1. Програма запитує в користувача, продовжувати роботу по натиснутій клавіші або ні. При введенні символів 'y', 'Y', 'n', 'N' – відбувається вихід з циклу, інакше на екран виводиться відповідне повідомлення.

```
#include <iostream>
using namespace std;

void main()
{
  char ans;
  cout << "Продовжити (Y/N)?";
  cin >> ans;
  while((ans!='Y')&&(ans!='y')&&(ans!='N')
        &&(ans!='n'))
  {
    cout << "\nНе правильно натиснута клавіша\n";
    cout << "Продовжити (Y/N)?";
    cin >> ans;
  }
}
```

Приклад 2. Підрахувати кількість символів у слові.

```
#include <iostream>
using namespace std;

void main()
{
  char name[15];
  int count = 0;
  cin >> name;
  while (name[count] != 0)
    count++;
  cout << count;
}
```

Цикл може бути організований як цикл з післяумовою **do … while**
Синтаксис:

do
{
блок з одного або більш операторів;
}
while (<вираз>);

Цикл має мінімум одну ітерацію. Цикл продовжується доти, поки вираз істинний, тобто не дорівнює 0 (false). Якщо **блок** складається з одного оператора, фігурні дужки можна не ставити.

Приклад, аналогічний прикладу 1:

```
#include <iostream>
using namespace std;

void main()
{
  char ans;
  do
  {
    cout <<"\nБудете продовжувати введення ( Y/N )?";
    cin >> ans;
  }
  while ((ans!='Y')&&(ans!='y')&&(ans!='N')
           &&(ans!='n'));
}
```

У С++ передбачена можливість дострокового виходу з програми до її нормального завершення за допомогою функції **exit()**.

Синтаксис:

exit(status) - передається значення операційній системі через оточення **errorlevel** – може перевірятися в командних файлах.

Як і в арифметичному циклі **for**, в циклах **while**, **do…while** мають місце оператор **break** для виходу з циклу та оператор **continue** для повернення на початок циклу.

6.3. Оператори switch та goto

Оператор **switch** замінює короткий оператор, що розгалужується **if… else**

Синтаксис:

```
switch ( вираз )
{
    case <конст_вираз 1>:
блок операторів С++;
    case  <конст_вираз 2>:
блок операторів С++;
    ...
    case  < конст_вираз n>:
       блок операторів С++;
    [default:
       блок операторів С++;]
}
```

Після обчислення виразу в заголовку оператора, його результат послідовно порівнюється з константами-вираженнями **<конст_вираз 1> ... <конст_вираз n>** починаючи із самого верхнього, поки не буде встановлена

їхня відповідність. Тоді виконуються оператори усередині відповідного **case**, і після цього управління переходить на такий константний вираз без перевірок. Саме тому наприкінці кожної послідовності операторів повинен бути присутнім оператор **break**, що забезпечує вихід з оператора **switch**. Гілка, яка має назву **default** (по умовчанню) може бути відсутня. Якщо вона є, то послідовність операторів, що знаходиться за словом **default**, виконується тільки тоді, коли порівняння з жодної зі стоячих вище константних виразів не є істинним.

Приклад:

```
switch (t)
{
  case 1:
    cout << "\nПовідомлення 1"; break;
  case 2:
    cout << "\nПовідомлення 2"; break;
//      ...
  default:
    cout << "\nЗрадливий код";
}
```

Якщо **break** не ставиться наприкінці блока операторів, і ми потрапляємо за умовою в блок, то здійснюється «провал донизу».

Приклад:

```
switch (num)
{
  case 1:
    cout << "1";
  case 2:
    cout << "22";
  case 3:
    cout << "333";
}
```

Якщо **num** дорівнює **1**, то результат буде:

```
122333
```

Без використання оператору **break** можуть бути оригінальні рішення.

Приклад:

```
switch (num)
{
  case 1:
  case 2:
  case 3:
    cout << "Номер від 1 до 3";
    break;
```

```
    case 4:
      cout << "Номер 4";
      break;
    default:
      cout << "Помилковий номер";
}
```

У загальному випадку <**виразом**> може бути будь-який вираз, константа або цілочисельна змінна. Константи-вираження – це константи цілочисельного типу.

Оператор **switch** зручно використовувати при реалізації меню.

Особливістю роботи з оператором switch є те, що в середині окремого блоку case не можна оголошувати змінні без використання фігурних дужок { }. Фігурні дужки визначають область видимості та існування окремої змінної. Тобто, для оголошення та використання змінної у блоці case необхідно взяти його у фігурні дужки.

Приклад обчислення функції аргументу x від заданого типу t:

```
#include <iostream>
using namespace std;

void main()
{
   int t;
   double x;
   cout<<"Введіть значення аргументу x ";
   cin>>x;
   cout<<"Введіть значення типу функції 1 або 2 ";
   cin>>t;
   switch (t)
   {
     case 1:
     {
       double y = x*x;
       cout << "\ny = " <<y;
       break;
     }
     case 2:
     {
       double y = x*x*x;
       cout << "\ny = " <<y;
       break;
     }
     default:
       cout << "\nНевірно заданий тип функції";
   }
}
```

Оператор **goto** рідко зустрічається, краще цим оператором не користуватися.

Приклад використання:

```
goto tt;  // Перехід на мітку tt.
c= 12;
...
tt: ...
```

Контрольні питання

1. Чим відрізняється арифметичний цикл від ітераційного?
2. Розкрийте сутність арифметичного циклу for.
3. Розкрийте сутність ітераційного циклу з передумовою while.
4. Розкрийте сутність ітераційного циклу з післяумовою do…while.
5. Розкрийте сутність призначення операторів switch та goto.
6. Для чого потрібні оператори break та continue?
7. Що таке “вічний цикл”?
8. Наведіть приклад арифметичного циклу for.
9. Наведіть приклад ітераційного циклу з передумовою while.
10. Наведіть приклад ітераційного циклу з післяумовою do…while.

Завдання

Арифметичний цикл

Знайти значення суми або добутку. Значення змінних вибрати самостійно

Приклад виконання завдання.

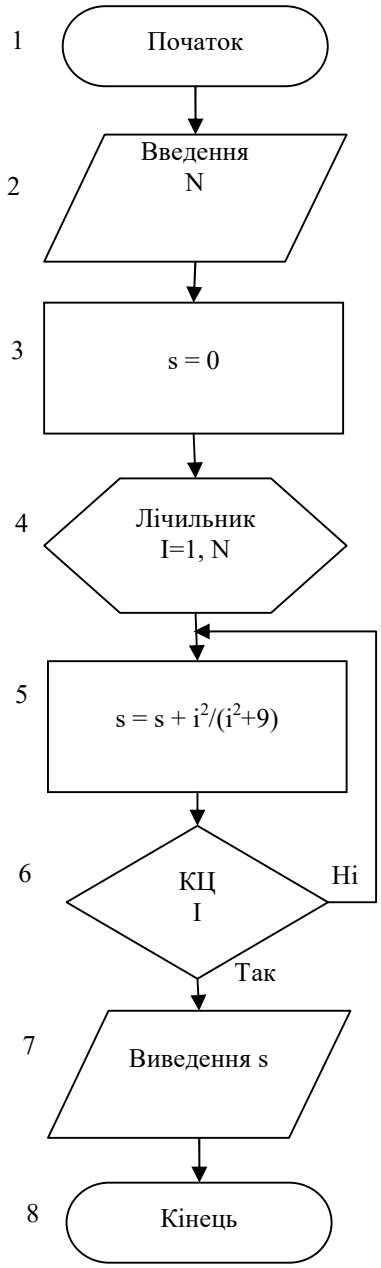
Знайти значення суми $\sum_{i=1}^{n} \frac{i^2}{i^2+9}$.

Код програми має наступний вигляд:

```
#include <iostream>
using namespace std;

void main()
{
 int i,n;
 double s;

 cout<<"Введіть n ";
 cin>>n;

 s = 0;
 for(i = 1; i<=n; i++)
      s+=(1.*i*i)/(i*i+9.);
```

```
 cout<<"s = "<<s;
}
```

Наведемо блок-схему вирішення даної задачі.



Варіанти індивідуальних завдань

1. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n} \frac{i^2 + 2i}{2^i}$$

2. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$S = \sum_{i=1}^{n} \left(i^2 - 1\right)\left(i^3 + 1\right) + in$$

3. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$C = \prod_{i=0}^{n-1} \frac{n-i}{n+i}$$

4. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sqrt{8 \sum_{k=1}^{n} \frac{1}{(2k-1)^2}}$$

5. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{k=0}^{n} (a + kn)$$

6. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \prod_{i=1}^{n} \left(i^2 + \frac{1}{e^i} \right)$$

7. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = 4 \sum_{k=1}^{n} \frac{1}{k(k+1)(k+2)}$$

8. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = 3 \sum_{k=1}^{n} \frac{x}{(2k-1)(2k+1)}$$

9. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sqrt{\sum_{i=0}^{n} \frac{1}{2^i}}$$

10. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n} n \left(i^3 + \frac{5}{i} \right)$$

11. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \sqrt{6 \sum_{k=1}^{n} \frac{1}{k^2}}$$

12. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \prod_{k=1}^{n} \frac{1}{k^2 + 1}$$

13. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \prod_{k=1}^{n} \frac{2k-1}{2k+2}$$

14. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \prod_{i=1}^{n} \frac{xi}{2+i}$$

15. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n} \frac{x + \cos(ix)}{2^i}$$

16. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \sum_{i=1}^{n} \frac{x}{i(i+1)}$$

17. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$S = \sum_{i=0}^{n} \frac{x}{4^i + 5^{i+2}}$$

18. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$H = \sum_{k=1}^{n} \frac{a^2}{(2k+1)^2}$$

19. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \prod_{i=1}^{n} \frac{i+1}{i+2}$$

20. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sqrt{18 \sum_{k=1}^{n} \frac{1}{k^3}}$$

21. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \prod_{k=1}^{n} \left(\frac{k}{k+1} - \cos^k |x| \right)$$

22. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$S = 1 + \sum_{k=1}^{n} \frac{1}{2k}$$

23. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \sum_{i=1}^{n} \frac{1}{(2i)^2}$$

24. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \prod_{i=1}^{n} \frac{b^2}{i^2 + 2i + 3}$$

25. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \prod_{k=1}^{n} \frac{b + 0{,}5(k^2 - 4k)}{n}$$

26. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sqrt[3]{10 \sum_{k=1}^{n} \frac{1}{(2k+1)^2}}$$

27. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$S = \prod_{i=1}^{n} \frac{nx^2 + 3}{2^i}$$

28. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{k=1}^{n} \frac{(-1)^k}{(2k+1)k}$$

29. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{i(i+1)}$$

30. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n} \frac{(-x)^{i+1}}{i(i+1)(i+2)}$$

Вкладені цикли

Знайти значення суми або добутку. Значення змінних вибрати самостійно.

Варіанти індивідуальних завдань

1. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{k=0}^{n} \frac{(-1)^k (k+1)}{k!}$$

2. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$S = \sum_{i=1}^{n} \sum_{j=1}^{i} \frac{1}{j+3i}$$

3. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$\tilde{N} = \sum_{i=1}^{n} (i! + n)$$

4. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{k=1}^{n} \frac{1}{k!}$$

5. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = n! + \sum_{k=1}^{n} \sqrt{a + \frac{1}{k}}$$

6. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=0}^{n} \frac{(-2)^i}{i!}$$

7. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \prod_{k=1}^{n} \sum_{i=1}^{k} \frac{i^2}{n}$$

8. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \prod_{k=1}^{n} \left(1 + \frac{k^2 \cdot x}{k!}\right)$$

9. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n} \frac{1}{i!}$$

10. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n} \frac{(-1)^i}{i!}$$

11. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \prod_{k=1}^{n}\left(1 + \frac{\sin(kx)}{k!}\right)$$

12. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \prod_{k=1}^{n}\left(2 + \frac{1}{k!}\right)$$

13. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{k=1}^{m}\prod_{i=1}^{k}\frac{i+1}{i^2 - 0{,}5}$$

14. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \prod_{i=1}^{n}\left(x - \frac{1}{i!}\right)^2$$

15. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n}\frac{x^i + 2n}{i!}$$

16. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \prod_{i=1}^{n}\frac{(1-x)^{i+1} + 1}{((i-1)!+1)^2}$$

17. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$S = \sum_{i=1}^{n}\left(\frac{1}{i!} + \sqrt{|x|}\right)$$

18. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$H = 1 + \sum_{k=1}^{n}\frac{a^k}{k!}$$

19. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n}\frac{1}{i!+3}$$

20. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{k=1}^{n}(k!) - 2n!$$

21. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = 1 + \sum_{k=1}^{n} \frac{x^k}{k!}$$

22. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$S = \sqrt[5]{n! \sum_{i=1}^{n} \frac{1}{i^2}}$$

23. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$P = \prod_{i=2}^{n} \left(1 - \frac{1}{i!}\right)$$

24. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n} \frac{(-2)^i}{i!} \lg(b)$$

25. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{k=1}^{n} \frac{(-1)^k}{k!} \cdot \frac{b}{n}$$

26. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{k=1}^{n} \frac{(-1)^k}{k!}$$

27. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$S = \sum_{i=1}^{n} \frac{x}{i!} + 2 \cdot \sqrt{|x|}$$

28. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \prod_{k=1}^{n} \frac{e^{n+5}}{k!}$$

29. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$Y = \sum_{i=1}^{n} \sum_{j=1}^{i} \frac{1}{2j + i}$$

30. Скласти програму та блок-схему алгоритму циклічного обчислювального виразу:

$$S = \sum_{i=1}^{n} \frac{x^{i+2}}{i!+4}$$

Ітераційний цикл

Знайти значення суми або добутку. Значення змінних вибрати самостійно.

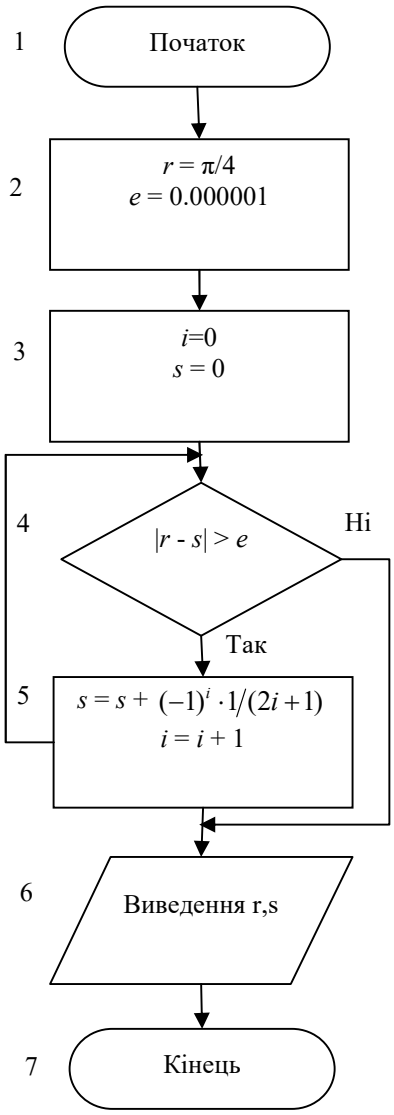
Приклад виконання завдання.

Знайти суму ряду $\sum_{i=0}^{\infty}(-1)^i \cdot \frac{1}{2i+1}$.

Код програми має наступний вигляд:

```
#include <iostream>
#include <cmath>
using namespace std;

#define M_PI        3.14159265358979323846

void main()
{
 int i;
 double s,r,e;

 r = M_PI/4;
 e = 0.000001;

 i = 0;
 s = 0;
 while(fabs(r-s)>e)
 {
  s+=pow(-1.,i)/(2.*i+1);
  i++;
 }

  cout<<"\ns = "<<s;
  cout<<"\nr = "<<r;
}
```

Наведемо блок-схему вирішення даної задачі.



Результат виконання програми:

s = 0.785397

r = 0.785398

Варіанти індивідуальних завдань

Знайти суму ряду:

1. $\sum_{n=0}^{\infty}(-1)^n \frac{1}{n!} = 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + ... = \frac{1}{e}$;
2. $\sum_{n=0}^{\infty}\frac{1}{n!} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + ... = e$;
3. $\sum_{n=1}^{\infty}(-1)^{n-1} \frac{1}{n!} = 1 - \frac{1}{2!} + \frac{1}{3!} - \frac{1}{4!} + ... = \ln 2$;
4. $\sum_{n=0}^{\infty}\frac{1}{2^n} = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... = 2$;

5. $\sum_{n=0}^{\infty}(-1)^n \frac{1}{2^n} = 1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \ldots = \frac{2}{3}$;
6. $\sum_{n=0}^{\infty}(-1)^n \frac{1}{2n+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots = \frac{\pi}{4}$;
7. $\sum_{n=1}^{\infty}\frac{1}{n(n+1)} = \frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \frac{1}{3\cdot 4} + \ldots = 1$;
8. $\sum_{n=1}^{\infty}\frac{1}{(2n-1)(2n+1)} = \frac{1}{1\cdot 3} + \frac{1}{3\cdot 5} + \frac{1}{5\cdot 7} + \ldots = \frac{1}{2}$;
9. $\sum_{n=2}^{\infty}\frac{1}{(n-1)(n+1)} = \frac{1}{1\cdot 3} + \frac{1}{2\cdot 4} + \frac{1}{3\cdot 5} + \ldots = \frac{3}{4}$;
10. $\sum_{n=1}^{\infty}\frac{1}{(4n-1)(4n+1)} = \frac{1}{3\cdot 5} + \frac{1}{7\cdot 9} + \frac{1}{11\cdot 13} + \ldots = \frac{1}{2} - \frac{\pi}{8}$;
11. $\sum_{n=1}^{\infty}\frac{1}{n(n+1)(n+2)} = \frac{1}{1\cdot 2\cdot 3} + \frac{1}{2\cdot 3\cdot 4} + \ldots = \frac{1}{4}$;
12. $\sum_{n=1}^{\infty}\frac{1}{n^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots = \frac{\pi^2}{6}$;
13. $\sum_{n=1}^{\infty}(-1)^n \frac{1}{n^2} = 1 - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \ldots = \frac{\pi^2}{12}$;
14. $\sum_{n=1}^{\infty}\frac{1}{(2n-1)^2} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + \ldots = \frac{\pi^2}{8}$;
15. $\sum_{n=1}^{\infty}\frac{1}{n^4} = 1 + \frac{1}{2^4} + \frac{1}{3^4} + \ldots = \frac{\pi^4}{90}$;
16. $\sum_{n=1}^{\infty}(-1)^{n-1} \frac{1}{n^4} = 1 - \frac{1}{2^4} + \frac{1}{3^4} - \ldots = \frac{7\pi^4}{720}$;
17. $\sum_{n=1}^{\infty}\frac{1}{(2n-1)^4} = 1 + \frac{1}{3^4} + \frac{1}{5^4} + \ldots = \frac{\pi^4}{96}$;
18. Обчислити корінь квадратний $x = \sqrt{a}$ по алгоритму Герона $x_{n+1} = x_n + \frac{1}{2}\left(\frac{a}{x_n} - x_n\right)$, де $n = 0, 1, 2\ldots$, прийнявши початкове значення $x_0 = 1$; розрахунок зупинити, якщо $|x_{n+1} - x_n| \le \varepsilon$.
19. Обчислити $y = \sin x$ двома способами: використовуючи стандартну функцію та формулу розкладання у ряд

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = \sum_{k=0}^{n}\frac{(-1)^k x^{2k+1}}{(2k+1)!}.$$

Порівняти два отриманих результати та отримати їх абсолютну різницю. Врахувати введення аргументів у градусах.

20. Обчислити $y = \cos(x)$ двома способами: використовуючи стандартну функцію та формулу розкладання у ряд

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots = 1 + \sum_{k=1}^{n} (-1)^k \frac{x^{2k}}{(2k)!} .$$

Порівняти два отриманих результати.

Суміщення розгалуженого та циклічного процесів

Визначити значення функції. Значення змінних вибрати самостійно.

Варіанти індивідуальних завдань

1. $$F = \begin{cases} \prod_{k=1}^{n} \left(1 + \frac{k^2 \cdot x}{k!}\right), & \text{якщо} \quad x > 4,\ \text{n} > 0 \\ x\sqrt{b^3}, & \text{якщо}\ \ 0 < x < 4,\ b > 0 \\ 0 \quad - & \text{в інших випадках} \end{cases}$$

2. $$D = \begin{cases} \sum_{x=1}^{m} e^{x+1}, & \text{якщо } m > 5,\ 0 < \text{x} < 12 \\ \sum_{x=1}^{m} \frac{1}{x}, & \text{якщо m - целое, x} \neq 0 \\ 0 \quad - & \text{в інших випадках} \end{cases}$$

3. $$Y = \begin{cases} \sum_{i=1}^{x} \frac{e^{x-1} + \ln x}{i}, & \text{якщо } x > 0,\ x - \text{ціле} \\ \sum_{i=1}^{5} e^{\frac{x}{i}}, & \text{в інших випадках} \end{cases}$$

4. $$C = \begin{cases} x \cdot e^{x^2-1}, & \text{якщо } x < 10 \\ \pi \cdot \sum_{i=1}^{x} e^i, & \text{якщо x - ціле, x} > 1 \\ 0 \quad - & \text{в інших випадках} \end{cases}$$

5. $$R = \begin{cases} \sum_{i=1}^{x} \lg i, & \text{якщо x} > 0 \text{ та ціле} \\ \sum_{\text{i}=0}^{10} \frac{(-1)^i \cdot (i+1)}{i!} & \text{в інших випадках} \end{cases}$$

6. $$Y = \begin{cases} \sum_{i=1}^{n} \frac{1}{i^5}, & \text{якщо n - ціле, тт } \text{n} < 10 \\ \prod_{i=1}^{10} \left(2 + \frac{1}{i!}\right) & - \text{ в інших випадках} \end{cases}$$

7. $$L=\begin{cases}\sum\limits_{i=1}^{5}\left(e^{x-1}+\ln\dfrac{x}{i}\right), & \textit{якщо } \text{x} > 0 \text{ тт кратне } 2\\ \sum\limits_{\text{i-1}}^{50}\dfrac{1}{\text{i}^3} & - \quad \textit{в інших випадках}\end{cases}$$

8. $$P=\begin{cases}\sum\limits_{i=0}^{15}\dfrac{a}{4^i+5^{i+2}}, & \textit{якщо } 2 < \text{a} < 100\\ \prod\limits_{\text{i=2}}^{10}\left(1-\dfrac{1}{\text{i!}}\right) & - \quad \textit{в інших випадках}\end{cases}$$

9. $$Y=\begin{cases}\sum\limits_{k=1}^{n}\dfrac{1}{k^5}, & \textit{якщо } \text{n - ціле, } 1 < \text{n} < 15\\ \prod\limits_{\text{i=1}}^{10}\left(2+\dfrac{1}{\text{i!}}\right) & - \quad \textit{в інших випадках}\end{cases}$$

10. $$G=\begin{cases}\sum\limits_{i=1}^{5}\left(e^{x-1}+\ln\dfrac{x}{i}\right), & \textit{якщо } x > 0 \text{ та кратне } 2\\ \sum\limits_{i=1}^{50}\dfrac{1}{i^3} & - \quad \textit{в інших випадках}\end{cases}$$

11. $$Y=\begin{cases}\sum\limits_{i=1}^{n}\dfrac{(-1)^i}{i!}\cdot\dfrac{b}{z}, & \textit{якщо } \text{n} > 0 \text{ тт ціле, b} > 4, \text{z} > 1\\ \prod\limits_{i=1}^{10}\left(2+\dfrac{1}{i!}\right) & - \quad \textit{в інших випадках}\end{cases}$$

12. $$R=\begin{cases}\sum\limits_{i=1}^{x}\dfrac{(-1)^i}{i!}, & \textit{якщо } x > 0 \text{ та ціле}\\ \prod\limits_{i=1}^{5}\dfrac{x^2-e}{i} & - \quad \textit{в інших випадках}\end{cases}$$

13. $$P=\begin{cases}\prod\limits_{i=1}^{n}\dfrac{i}{2+i}+\sum\limits_{i=1}^{n}i^2, & \textit{якщо } n > 0 \text{ та ціле}\\ 0, & \textit{якщо } x = 0\\ \pi\cdot x\sum\limits_{i=1}^{5}\dfrac{1}{i^2} & - \quad \textit{в інших випадках}\end{cases}$$

14. $$Y=\begin{cases}\sum\limits_{k=1}^{n}(-1)^k\left(2k^2+1\right), & \textit{якщо } n > 0 \text{ та ціле}\\ \sum\limits_{i=1}^{50}\sum\limits_{j=1}^{i}\dfrac{j-i+1}{i+j} & - \quad \textit{в інших випадках}\end{cases}$$

15. $$F=\begin{cases}\sum\limits_{i=1}^{n}\dfrac{1}{i!}+\sqrt{|x|}, & \textit{якщо } \text{n - ціле тт } 2 \le \text{n} \le 12\\ \sum\limits_{i=1}^{25}\dfrac{1}{(2i)^2} & - \quad \textit{в інших випадках}\end{cases}$$

16. $K = \begin{cases} xe^x, & \textit{якщо } \text{x} \le 0 \\ \dfrac{1}{\pi}\displaystyle\sum_{i=1}^{6}\frac{i!}{x} & - \textit{ в інших випадках} \end{cases}$

17. $Y = \begin{cases} \displaystyle\prod_{k=1}^{n}\left(\frac{k}{k+1} - b^k|x|\right), & \textit{якщо } \text{n} > 0 \text{ тт ціле} \\ \displaystyle\sum_{k=1}^{10}\frac{x+\sqrt{|b|}}{2^k}, & \textit{якщо } \text{x} > 0 \text{ тт b кратне 2} \\ 0 & - \textit{ в інших випадках} \end{cases}$

18. $P = \begin{cases} \displaystyle\sum_{k=1}^{n}\frac{(-1)^{k+1}}{k(k+1)}, & \textit{якщо } \text{n} > 0 \text{ тт кратне 3} \\ C\displaystyle\sum_{i=1}^{4}0.3i & - \textit{ в інших випадках} \end{cases}$

19. $F = \begin{cases} \displaystyle\prod_{k=1}^{n}\frac{(1-x)^{k+1}+1}{((k-1)!+1)^2}, & \textit{якщо } \text{n} > 0, \text{ ціле тт кратне 2} \\ 0 & - \textit{ в інших випадках} \end{cases}$

20. $A = \begin{cases} \displaystyle\prod_{i=1}^{n}\frac{e^{x+5}}{i!}, & \textit{якщо } 1 \le \text{x} \le 5 \\ \dfrac{1}{\text{e}^{\text{x}}+5} & - \textit{ в інших випадках} \end{cases}$

21. $Y = \begin{cases} \displaystyle\sum_{i=1}^{n}\frac{x^i}{i!}, & \textit{якщо } \text{n} > 0 \text{ тт ціле, } \text{x} < 0 \\ 0.5\lg x, & \textit{якщо } \text{x} > 0 \\ 0 & - \textit{ в інших випадках} \end{cases}$

22. $F = \begin{cases} \displaystyle\sum_{k=1}^{n}\frac{(-1)^{k+1}}{k(k+1)}\cdot x, & \textit{якщо } \text{x} < 0, \text{ n} > 0 \textit{ та ціле} \\ \displaystyle\sum_{k=1}^{10}k^3 \cdot \sum_{l=1}^{15}(k-l)^2 & - \textit{ в інших випадках} \end{cases}$

23. $H = \begin{cases} \lg(x-a), & \text{яякщ x} > 0 \text{ тт a} \le 0 \\ \text{a}\displaystyle\sum_{i=1}^{5}\left(\text{e}^{\text{x}+1}\right)\cdot x^2 & - \textit{ в інших випадках} \end{cases}$

24. $L = \begin{cases} \dfrac{1}{x}\cdot\displaystyle\sum_{i=1}^{15}i!, & \textit{якщо } \text{x} > 0 \\ \displaystyle\sum_{i=1}^{25}\sum_{j=1}^{i}\frac{1}{j+3i} & - \textit{ в інших випадках} \end{cases}$

25. $$Y=\begin{cases}\prod\limits_{i=1}^{n} e^{i}, & \textit{якщо } \mathrm{n} \ge 2 \text{ тт ціле}\\ \sum\limits_{i=1}^{4}\left(\lg\dfrac{x}{i}+\ln\dfrac{x}{i}\right) & - \quad \textit{в інших випадках}\end{cases}$$

26. $$F=\begin{cases}y^{x+z}\cdot e^{y}, & \textit{якщо } \mathrm{y} \le -2\\ \sum\limits_{\mathrm{i}=1}^{4} \mathrm{i}\cdot\lg(\mathrm{y}+2) & - \quad \textit{в інших випадках}\end{cases}$$

27. $$Y=\begin{cases}\pi\cdot\prod\limits_{i=1}^{z}\lg z, & \textit{якщо } 1 \le \mathrm{z} \le 5 \text{ тт z - ціле}\\ \sum\limits_{\mathrm{i}=1}^{30}\sum\limits_{\mathrm{j}=1}^{i}\dfrac{1}{2j+i} & - \quad \textit{в інших випадках}\end{cases}$$

28. $$H=\begin{cases}\ln\left(x+\prod\limits_{i=1}^{10}\lg(ix)\right), & \textit{якщо } \mathrm{x} > 0\\ \sum\limits_{\mathrm{i}=1}^{15}\mathrm{e}^{\mathrm{x}} i & - \quad \textit{в інших випадках}\end{cases}$$

29. $$Y=\begin{cases}\sum\limits_{i=1}^{m}\dfrac{(-1)^{i}}{i!}, & \textit{якщо } \mathrm{m} > 0 \text{ тт ціле}\\ \sum\limits_{\mathrm{i}=2}^{50}\dfrac{\mathrm{i}+1}{\mathrm{i}+2} & - \quad \textit{в інших випадках}\end{cases}$$

30. $$Y=\begin{cases}\sum\limits_{i=1}^{n}\dfrac{x^{2}+\lg i}{i^{2}}, & \textit{якщо } n > m,\ \mathrm{n} > 0\\ \sum\limits_{i=1}^{m}\dfrac{\ln x+e^{i}}{xi!}, & \textit{якщо } m > n,\ \mathrm{m} > 0\\ 0 & - \quad \textit{в інших випадках}\end{cases}$$

РОЗДІЛ 7. ЗОВНІШНІ ПРИСТРОЇ ТА СИМВОЛЬНЕ ВВЕДЕННЯ/ВИВЕДЕННЯ. РЯДКОВІ, ЧИСЛОВІ ФУНКЦІЇ ТА ФУНКЦІЇ РОБОТИ З ДАТОЮ ТА ЧАСОМ

7.1. Загальна концепція та функції символьного введення-виведення

На відміну від інших мов програмування, C++ не має команд введення-виведення. Мова C++ характеризується високою переносимістю: це означає, що програма, яка компілюється і виконується на одному комп'ютері, може компілюватися та виконуватися на інших з мінімальними модифікаціями. Розроблювачі компілятора також пишуть функції введення/виведення для кожного типу машини. Тому, коли програма написана на C++ виводить на екран символ, то вона буде працювати незалежно від того з якою операційною системою вона має справу: DOS, WINDOWS або UNIX.

Використання потоків введення-виведення дозволяє використовувати ті ж самі функції для введення, як з клавіатури, так і з модему. Ви можете використовувати ті ж функції для запису файлу на диск, виведення на принтер або на екран (табл. 7.1).

Таблиця 7.1

Стандартні пристрої в C++

Опис	Ім'я C++
Екран	stdout
Клавіатура	stdin
Принтер	stdprn
Послідовний порт	stdaux
Повідомлення про помилки	stderr

Форматоване введення-виведення на принтер.

Виведення на принтер результатів виконання програми легко здійснити за допомогою об'єктів **ofstream** і його членів-функцій **ofstream device (device_name)**

Увага!!! ofstream використовує заголовний файл fstream або ostream.

Приклад: Програма запитує в користувача ім'я і прізвище, а потім друкує прізвище та ім'я на принтер.

```
#include <iostream>
#include <fstream>
using namespace std;

void main()
{
 ofstream prn("LPT1");
 ifstream ek("CON");
```

```
 char first[20];
 char last[20];
 cout << "Ввести прізвище ";
 cin >>first;
 cout << "Ввести ім'я ";
 ek >> last;
 prn << first << "\n" << last  <<"\n";
}

//Виведення на принтер
ofstream prn("LPT1");
prn << first << "\n" << last  <<"\n";
```

Функції символьного введення-виведення

Об'єкти **cin** і **cout** – оператори форматованого введення-виведення і не є функціями символьного введення-виведення. Вони призначені тільки для введення з клавіатури та виведення на екран. Більш універсальними є функції символьного введення-виведення, що дозволяють працювати з будь-якими пристроями.

Функції get() і put().

get() – вводить один символ із стандартного пристрою введення (якщо не було перепризначень, то це клавіатура).

put() – виводить одиничні символи в стандартний пристрій виведення (якщо не було перепризначень, то це екран).

Синтаксис:

device. get(char var)

device – будь-який стандартний пристрій

Якщо введення даних виконується з клавіатури, можна використовувати **cin**. Якби введення здійснювалося з модему, то необхідно було б використовувати об'єкт **ifstream**.

Синтаксис функції **put():**

device. put(char var)

Приклад:

Наступна програма по одному символу зафарбовує ініціали користувача. Програма повинна викликати 2 функції **get()** для кожного введеного символу. Після того, як на запрошення функції **get()** ви наберете символ і натиснете <Enter>, C++ знайде дані у вигляді потоку з 2-х символів. Функція **get()** спочатку одержує набрані символи, а потім одержує символ '\n' (клавіша <Enter>).

Приведемо текст програми.

```
#include <iostream>
using namespace std;

void main()
{
 char a,t,l;
 cout << "Ввести 1-й ініціал ";
 cin.get(a);
 t=a;
 cin.get(a); // Ігнорується символ переведення
             // рядка
 cout << "Ввести 2-й ініціал ";
 cin.get(a);
 l=a;
 cin.get(a); //Ігнорується символ переведення
             //рядка
 cout << "\nІніціали: \n";
 cout.put(t);
 cout.put(l);
}
```

Результат виконання програми:

```
  Ввести 1-й ініціал? A
  Ввести 2-й ініціал? C
 Ініціали:
 AC
```

Приклад: за допомогою функції **get()** заповнюється масив, що містить введений користувацький рядок.

Увага: оператор **cin** обмежений, введення рядків здійснюється по одному слову. Використовуючи **get()**, можна створювати власні функції, у котрих немає обмежень.

```
#include <iostream>
using namespace std;

#define MAX 25

int get_in_str(char str[], int len)
{
  int i = 0;
  char input_char;
  cin.get(input_char);
  while (i < (len - 1) && (input_char != '\n'))
  {
    str[i] = input_char; i++;
```

```
    cin.get(input_char);
  }
  str[i] = '\0';
  return i;
}

void main()
{
  int k, i;
  char input_str[MAX];
  cout << "Введіть повне ім'я ";
  k = get_in_str(input_str,MAX);
  for(i = 0; i < k; i++)
    cout.put(input_str[i]);
}
```

Функція **get()** – функція буферизованого введення; означає, що введені символи потрапляють у буфер, а потім у програму. Коли натискається клавіша <Enter>, вміст буфера передається в програму.

Приклад: Введення рядка символів (включаючи пробіли, символи табуляції і переведення рядка). Закінчення введення – клавіша <Ctrl+Z>.

```
#include <iostream>
using namespace std;

#define MAX 20

void main()
{
  char d[MAX];
  int i = 0;
  while(i < MAX - 1 && cin.get(d[i])) i++;
  d[i] = 0;
  cout << "------------------\n" << d ;
}
```

Для безпосереднього введення даних, тобто без буфера використовуються функції **getch()** і **putch()**.

Функції getch() і putch().

Вони схожі по формату на функції **get()** і **put()**. Функції **getch()** і **putch()** не буферизовані, тобто введенні символи не помітні на екрані. Якщо треба побачити символи на екрані, можна використовувати функцію **getche()**. Якщо треба одержати відповідь від програми відразу ж після введення з клавіатури, використовуйте **getch()**. Деякі програмісти не хочуть змушувати користувача натискати <Enter> у відповідь на запрошення або вибір із меню. Інші гадають,

що буферизоване введення дає користувачу можливість подумати, чи дійсно він хоче вибрати саме цю відповідь, тому що в цьому випадку до натискання клавіші <Enter> користувач може скористатися клавішею <Backspace> і відкоригувати рядок.

Прототипи функцій **getch()** і **putch()** описані в заголовному файлі **conio.h**.

Приклад**:**

Наступна програма ілюструє роботу функцій **getch()** і **putch()**. Видається запит користувачу ввести 5 букв, що за допомогою циклу записуються у масив **letters**. Символи не виводяться на екран, тому що **getch** виконує небуферизоване введення, програма одержує кожний символ, поміщає його в масив, і цикл повторюється, в той час як користувач вводить символи. (Якби це було буферизоване введення, цикл не почав би виконуватися доти, поки б не було натиснуто <Enter>.)

```
#include <iostream>
using namespace std;

#include <conio.h>

void main()
{
  int ctr;
  char letters[5];
  cout << " Введіть символи\n";
  for (ctr=0; ctr<5;ctr++)
    letters[ctr]=getch();

  for (ctr=0;ctr<5;ctr++)
    putch(letters[ctr]);
}
```

Функції gets() і puts()

Синтаксис:
char *gets(char *s);
int puts(char *s);

За допомогою функції **gets()** виконується зчитування символів із стандартного вхідного потоку **stdin**. Якщо зустрічається символ переходу на новий рядок, він замінюється символом кінця рядка ('\0'). При використанні функції **gets()** варто дотримуватись обережності. Адже число введених символів може перевищити розмір пам’яті, відведеної під рядок **s**. Функція **puts()** виводить рядок **s** у стандартний вихідний потік **stdout**.

Приклад:

```
#include <iostream>
using namespace std;
```

```
#define MAX 255

void main()
{
  char s[MAX];
  gets(s);
  puts(s);
}
```

Висновки:

Використання потоків форматованого введення-виведення, наприклад **cout; ofstream** дозволяє здійснити форматоване виведення на будь-який пристрій, включаючи принтер.

Хоча методи символьного введення-виведення можуть здаватися примітивними, проте, вони забезпечують гнучкість, тому що на їхній основі можна створювати власні функції введення-виведення.

7.2. Символьні функції

Прототипи символьних функцій описані в заголовному файлі <**ctype.h> (<cctype>)**. Якщо у програмі включено заголовний файл <**iostream**>, файл <**ctype.h> (<cctype>)** можна не включати.

isalpha(c) – Істина (не нуль), якщо буква
islower(c) – Істина (не нуль), якщо мала літера
isupper(c) – Істина (не нуль), якщо прописна буква
isdigit(c) – Істина (не нуль), якщо цифра від 0-9
isalnum(c) – Істина (не нуль), якщо цифра 0-9 або буква

Приклад:

Програма запитує в користувача його ініціал. Якщо користувач вводить що-небудь відмінне від букв, програма виводить повідомлення про помилку і пропонує повторити введення.

```
#include <iostream>
using namespace std;

void main()
{
  char initial;
  cout << "Введіть ініціал ";
  cin >> initial;
  while (!isalpha(initial))
  {
    cout <<"\n Помилка \n";
```

```
    cout << "Введіть ініціал ";
    cin >> initial;
  }
}
```

Функції перетворення символів.

tolower(c) – перетворить символ до нижнього регістра;
toupper(c) – перетворить символ до верхнього регістра.

7.3. Рядкові функції

Рядкові функції описуються у файлі <**string.h> (<cstring>)**. Якщо у програмі включено заголовний файл <**iostream**>, файл <**string.h> (<cstring>)** можна не включати.

char *strcat(char *s1, const char *s2);
додати рядок **s2** у кінець масиву символів **s1**. Масив **s1** повинен мати достатньо вільних елементів.
char *strncat(char *s1, const char *s2, unsigned m);
додати **m** символів рядка **s2** у кінець масиву символів **s1**. Масив **s1** повинен мати достатньо вільних елементів.
char *strchr(char *s, char c);
пошук першого входження символу **c** у рядку **s**
int strcmp(const char *s1, const char *s2);
порівнюються рядки **s1** і **s2**. Повертає:
якщо **s1 < s2 -** від'ємне значення;
якщо **s1 > s2 -** додатне значення;
якщо **s1** і **s2** збігаються – 0.
int strncmp(const char *s1, const char *s2, unsigned m);
теж, що і функція **strcmp**, але порівнюються перші **m** символів двох рядків.
Функції **stricmp** і **strnicmp** виконують ті ж дії, але без урахування регістра символів (тільки латинські символи).
char *strcpy(char *s1, const char *s2);
копіює рядок **s2** у рядок **s1**.
char *strncpy(char *s1, const char *s2, unsigned m);
копіює перші **m** символів рядка **s2** у рядок **s1**.
unsigned strlen(const char *s);
повертає довжину **s**. (Довжина рядка – кількість символів без урахування завершального нуля '\0'.)
char *strlwr(char *s);
переводить рядок **s** у нижній регістр (тільки латинські символи).
char *strupr(char *s);
переводить рядок **s** у верхній регістр (тільки латинські символи).
char *strset(char *s, int ch);
заповнює весь рядок **s** символами **ch**.

char *strnset(char *s, int ch, unsigned m);
замінює перші **m** символів рядка **s** на символ **ch**.
char *strrev(char *s);
перетворить рядок **s** в зворотному порядку.
unsigned strcspn(const char *s1, const char *s2);
повертає позицію в рядку **s1** символу з рядка **s2**.
char *strpbrk(char *s1, const char *s2);
повертає вказівку на перший символ рядка **s1,** який міститься в рядку **s2**.
char *strrchr(char *s, char c);
пошук останнього входження символу **c** у рядку **s**
unsigned strspn(const char *s1, const char *s2);
повертає позицію в рядку **s1** символа, що не входить у рядок **s2**.
char *strtok(char *s1, const char *s2);
виділяє фрагменти рядка **s1,** розділені одно- або багатосимвольними роздільниками з рядка **s2**. При першому звертанні до **strtok** видається адреса першого фрагмента. Наступні виклики з завданням **NULL** замість першого аргументу (**s1)** будуть видавати адреси наступних фрагментів з рядка **s1** доти, поки фрагментів не залишиться.

Функції перетворення рядків у числа.

int atoi(const char *s) – перетворює **s** до цілого (alphabetic to integer);
long atol(const char *s) – перетворює **s** до довгого цілого (alphabetic to long);
double atof(const char *s) – перетворює **s** до числа з плаваючої точкою (alphabetic to float).

7.4. Числові функції

Прототипом для роботи з числовими функціями є файл <**math.h.**> (**<cmath>**).

double ceil(double x) – повертає найближче більше ціле число.
double fabs(double x) – повертає абсолютне значення **x**.
double floor(double x) – повертає найближче менше ціле число.
double fmod(x,y) – повертає число з плаваючою точкою, що є залишком від ділення **x** на **y**, із тим же знаком, що і **x**. Аргумент **y** не може бути нулем.

На відміну від оператора обчислення модуля (**%**), що працює тільки з цілими числами, функція **fmod** дозволяє знаходити залишок від ділення чисел з плаваючою точкою.

double pow(double x, doubley) – повертає значення x^y; якщо **x <= 0**, то **y** повинен бути цілим. Якщо **x=0**, то **y** не може бути від'ємним.
double sqrt(double x) – повертає значення $\sqrt{x}$
double exp(double x) – повертає значення e^x
double log(double x) – повертає значення ln x
double log10(double x) – повертає значення lg x

Тригонометричні функції.

double cos(double x) → cos x
double sin(double x) → sin x
double tan(double x) → tg x
double acos(double x) → arccos x
double asin(double x) → arcsin x
double atan(double x) → arctg x
double atan2(double y, double x) → arctg (y/x)

При обчисленні тригонометричних функцій, кути (аргумент функцій **cos, sin** і **tan**, а також значення зворотних тригонометричних функцій **acos, asin, atan** і **atan2)** виражаються в радіанах.

Аргументи **x, y** і повертаємі значення числових функцій мають тип **double**.

Функції генерації випадкових чисел.

Заголовний файл <**stdlib.h> (<cstdlib>).**

int rand() – повертає псевдовипадкове ціле число в діапазоні від 0 до 2147483647.

Якщо потрібний інший набір псевдовипадкових чисел, використовується функція **srand(unsigned int seed)**, де **seed –** ціла зміна або константа.

Якщо необхідно при кожному запуску програми генерувати різноманітну множину випадкових чисел, можна використовувати наступний рядок:

```
srand(time(0)).
```

Для реалізації даного рядку необхідно включати файл <**time.h> (**<**ctime**>**)**.

7.5. Функції роботи з датою та часом

При розробці програмного забезпечення виникає необхідність роботи з датою та часом. Під датою будемо розуміти рік, місяць, день місяця. Під часом – години, хвилини, секунди та мілісекунди. У програмуванні виникають такі основні задачі роботи з датою, як отримання поточної дати та часу, арифметичні операції з датою та часом, використовуючи структури роботи з датою. Для роботи з датою та часом до головної програми необхідно включити хайдер **<time.h>**. Для розширеної роботи з функціями часу, а саме робота з мілісекундами, необхідно підключити хайдер **<sys\timeb.h>**.

Головна структура роботи з датою та часом – це структура tm.

```
struct tm
{
  int tm_sec;   /* Секунди */
  int tm_min;   /* Хвилини */
  int tm_hour;  /* Години (0--23) */
  int tm_mday;  /* День місяця (1--31) */
  int tm_mon;   /* Місяць (0--11) */
```

```
  int tm_year;  /* Рік(календарний рік мінус 1900) */
  int tm_wday;  /* День тижня (0--6; Неділя = 0) */
  int tm_yday;  /* День року (0--365) */
  int tm_isdst; /* відмінний від 0, якщо світло денне */
};
```

Елементи цієї структури можна проініціалізувати самостійно, або заповнити її поточною датою та часом, що приведено в наступному прикладі.

```
# include <iostream>
# include <ctime>
using namespace std;

void main()
{
   char str[50];
   time_t time1;
   struct tm *tm1;
   time(&time1);  // time1 кількість секунд з 1970 р.
   printf("%ld second\n",time1);
   tm1 =   localtime(&time1);
   printf("%d year\n",tm1->tm_year);
   printf("%d hour\n",tm1->tm_hour);
   printf("%d minutes\n",tm1->tm_min);
   printf("%d sec\n",tm1->tm_sec);

   strcpy(str, asctime(tm1));
   printf("%s\n", str);
}
```

У даному прикладі оголошується змінна ***t1*** типу **time_t**, що є еквівалентним типу даних **long**. Функція **time()**, у яку передається вказівка на структуру ***tm1*** (тип **tm**), повертає кількість секунд, починаючи з 1970-го року. Функція **localtime()** – генерує з цієї кількості секунд структуру ***tm1***.

Скажімо, якщо поточна дата – 27 січня 2023 року, а час 16:51:13, тоді буде мати місце наступний результат:

```
1674831073 second
123 year
16 hour
51 minutes
13 sec
Sun Jan 27 16:51:13 2023
```

Зауважимо, що рік розраховується відносно 1900 року, тому змінна year структури ***tm1*** дорівнюватиме 123.

Якщо необхідно отримати дату з часом у якості рядку (масиву символів **char**) треба використати функцію **asctime()**, входом у яку є структура типу **tm**.

Для отримання рядку з датою або з часом використовують функції, відповідно **char *_strdate(char*s1), char *_strtime(char*s1).**

У наступному прикладі виводиться поточна дата та час.

```
# include <iostream>
# include <ctime>
using namespace std;

void main()
{
char buf[15];
_strdate(buf);
cout<<"date "<<buf<<"\n";
_strtime(buf);
cout<<"time "<<buf<<"\n";
}
```

Існують моменти, коли треба деталізувати час, включаючи мілісекунди. Особливо це важливо, коли треба перевіряти ефективність за часом тих або інших операцій. З цією метою використовується структура **_timeb**.

```
struct _timeb
{
 long time; //кількість секунд починаючи з 1970 року
 short millitm; // кількість мілісекунд
 short  _timezone;  //  відмінність  у  хвилинах  за
                    Грінвічем та місцевим часом
 short dstflag; // відмінний від 0, якщо світло денне
};
```

Нижче приведемо приклад аналізу, який показує скільки часу треба на те, щоб пройшов порожній цикл з 1000000000 ітерацій. Даний приклад є дуже корисним для аналізу швидкодії алгоритмів, різноманітних технологій тощо.

```
# include <iostream>
# include <ctime>
using namespace std;

#include <sys\timeb.h>

void main()
{
   double nach, conec;
   struct _timeb timebuffer;

   _ftime( &timebuffer );
   nach = timebuffer.time+timebuffer.millitm/1000.0;

   for(long i=1;i<1000000000;i++);
```

```
    _ftime( &timebuffer );
    conec = timebuffer.time+timebuffer.millitm/1000.0;

    printf("Kol-vo1 %f sec\n",conec - nach);
}
```

Функція **_ftime()** у даному прикладі заповнює структуру ***timebuffer*** типу **_timeb**. Потім у змінну ***nach*** записуємо кількість секунд з 1970 року з врахуванням мілісекунд, та зчитуємо відповідну кількість секунд після циклу в 1000000000 ітерацій в змінну ***conec***. Різниця цих змінних і дасть нам кількість секунд на експериментальну операцію з урахуванням мілісекунд.

Якщо необхідно провести різноманітні операції з датами: віднімання дат, додавання певної кількості днів і т.і., треба перевести дати в секунди відносно 1970 року. Для цього призначена функція mktime.

Її синтаксис наступний.

time_t mktime(struct tm *t);

Для зворотного переведення секунд до структури **tm** використовується функція **localtime()** з наступним синтаксисом:

struct tm *localtime(const time_t *timer);

Контрольні питання

1. Як вивести дані на будь-який пристрій, наприклад принтер?
2. Які функції символьного введення/виведення ви знаєте?
3. Буферезовані та небуферезовані функції введення даних. Головні відмінності.
4. Які символьні функції для перевірки введеного значення вам відомі?
5. Які існують функції перетворення символів?
6. Перелічите основні рядкові функції.
7. Які ви знаєте функції перетворення рядків у числа?
8. Перелічите основні числові функції.
9. Перелічите функції генерації випадкових чисел.
10. Перелічите функції роботи з датою та часом.
11. Призначення структури tm.
12. Призначення структури _timeb.

Завдання

Скласти програму з використанням символьних функцій.

Приклад виконання завдання.

Підрахувати кількість слів у тексті та сформувати масив заданих слів.

Код програми має наступний вигляд:

```
#include <iostream>
using namespace std;

void main()
{
 char s[255];
 char mas_s[50][255];
 char*f;
 int i,kol;
 printf("Введите текст:\n");
 gets(s);
 kol = 0;
 f = strtok(s," ");


 while(f)
 {
  strcpy(mas_s[kol],f);
  kol++;
  f = strtok(0," ");
 }

  cout<<"\nkol = "<<kol;
  cout<<"\n\n";

  //вывод сформированного массива слов
  for(i = 0;i < kol;i++)
       cout<<mas_s[i]<<"\n";

}
```

У даному прикладі слід звернути увагу на символьну функцію strtok. Дана функція дозволяє виокремлювати окремі елементи рядку, розділені довільними символами. У нашому випадку треба виокремити слова, які можуть бути розділені довільною кількістю пропусків « ». Для першого звертання у функцію strtok передається рядок s:

```
f = strtok(s," ");
```

Вказівка f повертає вказівку на перше слово у рядку s, розділене пропусками, “ ”. Для того щоб виокремити інші слова в функцію strtok у якості першого параметру необхідно передати 0. Це означає, що надалі в рядку s будуть виокремлюватись слова доти, поки вказівка f не прийме нульового значення. Механізм є універсальним та призначений для виокремлення інформації на основі довільного шаблону. Виокремлення слів та запис їх до масиву відбувається наступним чином:

```
while(f)
{
 strcpy(mas_s[kol],f);
 kol++;
 f = strtok(0," ");
}
```

Варіанти індивідуальних завдань

1. У заданому тексті замінити слово A на слово B (довжини слів не співпадають).
2. У заданому тексті визначити кількість слів.
3. Маємо відомість, яка складається із 10 прізвищ. Знайти і надрукувати порядкові номера прізвищ, які задаються.
4. У слові “лірика” після кожного складу вставити склад “ма”.
5. У тексті забрати зайві пропуски між словами, залишивши по одному.
6. У тексті вставити між словами замість одного два пропуски.
7. У заданому слові переставити літери в алфавітному порядку.
8. У текстовій змінній "71D523CE8" поставити числа у порядку зменшення.
9. Розділити заданий текст на рядки. У формі розділення рядків використаний символ %.
10. У виразі "МІСЯЧНИЙ ШЛЯХ" переставити місцями слова.
11. Дано слово "АЛГОРИТМ". Вивести його на друк у зворотному порядку.
12. Скільки разів у тексті зустрічається задане слово? (Слова розділені пропусками).
13. У слові "ЕЛЕКТРОНІКА" переставити літери у порядку зворотному алфавіту.
14. Надрукувати найдовше слово із заданого тексту “Ціль статистики зосереджується в наданні фактів у найбільш стислій формі”.
15. У текстовій змінній "2C35IA4" представити числа у порядку зростання.
16. Існує відомість, яка складається із 10 прізвищ. Вивести на друк цю відомість у алфавітному порядку.
17. В аналізуючому слові всі голосні літери видалити та на їх місце поставити пропуск. Вивести на дисплей початкове слово та результат перетворень.
18. У слові "ЛІТЕРАТУРА" кожну голосну літеру виділити символом ".

19. У заданому слові змінити місцями першу і останню літери. Вивести на екран дисплея початкове слово і результат.

20. У заданому тексті видалити частину тексту , яка взята в дужки. (Разом з дужками).

21. Визначити процент літери "А" у слові "ЛІТЕРАТУРА".

22. У текстовій змінній "2С35ІА4" поставити числа у порядку зростання.

23. У заданому слові "БІЗНЕСМЕН" змінити місцями першу і останню літери.

24. Визначити, які символи і скільки разів зустрічаються у заданому слові "АЛЬТЕРНАТИВА".

25. Розшифрувати слово "НЕМСЕНЗІБ", переставити літери навпаки.

26. У слові "КОМУНІКАЦІЯ" розрахувати кількість приголосних літер.

27. Визначити літеру, на яку починається більше всього слів у тексті "Практикум починається на першому курсі , паралельно з читанням загального курсу".

28. Знайти найдовше слово заданого речення “Книга призначена для використання студентами та керівниками”.

29. У слові "РЕЧЕННЯ" усі приголосні літери замінити цифрами 1,2,3…

30. Існує відомість, яка складається з 10 прізвищ. Порахувати, скільки прізвищ закінчується на літеру "О".

31. У слові "ІНІЦІАТОР" після кожної голосної вставити таку ж літеру.

32. У слові "АЕРОМЕХАНІКА" розрахувати кількість голосних літер.

33. У слові "ЮРИСДИКЦІЯ" знайти і вивести на друк індекс останньої літери И.

34. Існує відомість, яка складається з 10 прізвищ. Вивести на друк цю відомість у порядку зворотному алфавіту.

35. У заданому слові визначити кількість літер А і замінити на літеру М.

РОЗДІЛ 8. ВКАЗІВКИ, ПОСИЛАННЯ ТА МАСИВИ

8.1. Вказівки

Змінні вказівки (або, як їх звичайно називають, просто вказівки) – це змінні, що містять адресу даних. Коли компілятор обробляє оператор визначення змінної, наприклад, int x = 5;, він виділяє пам'ять відповідно до типу (int) і ініціалізує її вказаним значенням (5). Всі звернення в програмі до змінної по її імені (x) замінюються компілятором на адресу області пам'яті, в якій зберігається значення змінної. Програміст може визначити власні змінні для зберігання адрес областей пам'яті. Такі змінні називаються вказівками. Отже, вказівки призначені для зберігання адрес областей пам'яті.

У C++ розрізняють три види вказівок – вказівки на об'єкт, на функцію та на void, що відрізняються властивостями і набором допустимих операцій. Вказівка не є самостійним типом, вона завжди пов'язана з яким-небудь іншим конкретним типом.

Вказівка на функцію містить адресу в сегменті коду, по якій розташовується виконуваний код функції, тобто адресу, по якій передається управління при виклику функції. Вказівки на функції використовуються для непрямого виклику функції (не через її ім'я, а через звернення до змінної, що зберігає її адресу), а також для передачі імені функції в іншу функцію як параметр. Вказівка функції має тип «вказівка функції, що повертає значення заданого типу і що має аргументи заданого типу»:

```
тип (*ім'я) ( список_типів_аргументів );
```

Наприклад, оголошення:

```
int (*fun) (double, double);
```

задає вказівку з ім'ям fun на функцію, що повертає значення типу int і що має два аргументи типу double.

Вказівка на об'єкт містить адресу області пам'яті, в якій зберігаються дані певного типу. Просте оголошення вказівки на об'єкт має вид:

```
тип *ім'я;
```

де тип може бути будь-яким, окрім посилання та бітового поля, причому тип може бути до цього моменту тільки оголошений, але ще не визначений.

Зірочка відноситься безпосередньо до імені, тому для того, щоб оголосити декілька вказівок, потрібно ставити її перед ім'ям кожного з них. Наприклад, в операторі

```
int *a, b, *c;
```

описуються дві вказівки на ціле з іменами **a** та **c**, а також ціла змінна **b**.

До роботи з вказівками відносяться два головних оператори:

***** оператор "значення за адресою";

& оператор "адреса значення".

Це бінарні оператори, що стоять перед змінною або виразом.

Розмір вказівки залежить від моделі пам'яті. Можна визначити вказівку на вказівку і так далі.

Вказівка на void застосовується в тих випадках, коли конкретний тип об'єкту, адресу якого потрібно зберігати, не визначений (наприклад, якщо в одній і тій же змінній в різні моменти часу потрібно зберігати адреси об'єктів різних типів). Вказівці на void можна привласнити значення вказівки будь-якого типу, а також порівнювати його з будь-якими вказівками, але перед виконанням яких-небудь дій з областю пам'яті, на яку він посилається, потрібно перетворити його до конкретного типу явним чином.

Вказівка може бути константою або змінною, а також вказувати на константу або змінну. Розглянемо приклади:

```
int x; // ціла змінна
const int cx = 1; // ціла константа
int *px ; // вказівка на цілу змінну
const int *pcx; // вказівка на цілу константу
int *const cp = &x; // вказівка-константа на цілу змінну
const int* const cpc = &cx; // вказівка-константа на цілу константу
```

У рядку `int *px` резервується місце для змінної-вказівки з ім’ям **px.** Якщо при оголошенні змінної перед ім’ям з’являється *****, то це означає, що вона оголошується як змінна-вказівка.

px – вказівка на ціле, тобто **px –** змінна, що містить адресу цілих значень. Якщо необхідно працювати з плаваючою точкою, то можна оголосити вказівку із відповідним типом, наприклад,

```
float *point;
```

Як видно з прикладів, модифікатор const, що знаходиться між ім'ям вказівки та зірочкою, відноситься до самої вказівки і забороняє її зміну, а const зліва від зірочки задає постійність значення, на яке вона вказує.

Для ініціалізації вказівок використовується операція отримання адреси &.

Ініціалізація вказівок

Вказівки найчастіше використовують при роботі із динамічною пам'яттю, яка називається купою (heap). Це вільна пам'ять, в якій можна під час виконання програми виділяти місце відповідно до потреб. Доступ до виділених ділянок динамічної пам'яті, званих динамічними змінними, проводиться тільки через вказівки. Час життя динамічних змінних – від точки створення до кінця програми або до явного звільнення пам'яті. У C++ використовується два способи роботи з динамічною пам'яттю.

При визначенні вказівки треба прагнути виконати її ініціалізацію, тобто присвоєння початкового значення. Ненавмисне використання неініціалізованих вказівок призводить до помилок в програмах.

Існує п’ять засобів завдання початкового значення змінній-вказівці:

1. Описати вказівку поза будь-якою функцією або оголосити її як **static**. Початковим значенням є нульова адреса пам'яті – 0. Перед тим, як почати користуватися вказівкою, варто зарезервувати пам'ять під значення.

2. Привласнити вказівці адресу змінної.

3. Привласнити вказівці значення іншої вказівки, до цього моменту вже правильно ініціалізованної.

4. Використовувати оператори динамічного розподілу пам'яті, такі як **new, delete** і **delete[]**.

5. Привласнити значення конкретної адреси (константи), при цьому необхідно привести константу до типу вказівки.

Приклади ініціалізації вказівок:

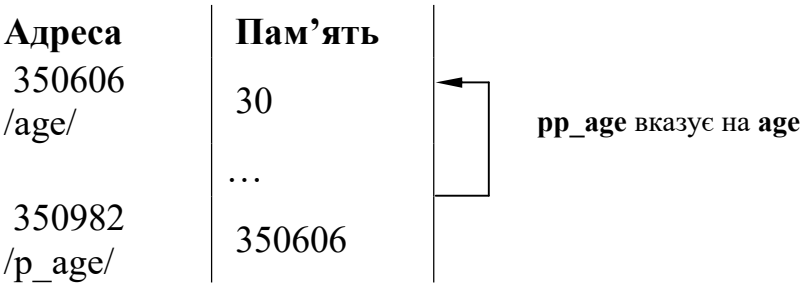
```
int *p1;

void main()
{
  static int *p2;

  int s = 10;
  int *p3 = &s;
//int *p3 (&s);

  int *p4,*p5;
  p4 = &s;
  p5 = p4;
}
```

Присвоювання адреси змінної **age** вказівці **p_age**

```
int age=30;
int *p_age;
p_age=&age;
```

Можна ініціалізовувати під час оголошення:

```
int age=30;
int *p_age=&age;
```



Для виведення значення змінної **age** можна використовувати 2 варіанта:

1) `cout << age;`
2) `cout << *p_age;`

Оператор * ставиться перед змінними вказівками тільки в двох випадках:

1. При оголошенні змінної-вказівки.
2. Для отримання значення, на яке вона вказує.

Після оголошення вказівки на конкретний тип об’єкта C++, спроба використання цієї вказівки для посилання на інший тип призведе до помилки:

```
void main()
{
  int* k;
  int j;
  char c;

  j = 5;
  c = 'C';
  k = &j;     //це припустимо
  k = &c;     //це призведе до помилки
}
```

Існує спеціальний тип вказівки **void*** на об’єкти C++ будь-якого типу, що виключає помилки компілятора, що стосуються вказівок. Вказівки типу **void*** можна використовувати в будь-яких цілях:

```
void main()
{
  void* k;
  int j;
  float f;
  char c;

//тут усе правильно
  k = &j;
  k = &f;
  k = &c;
}
```

Вказівки **void** мають важливу властивість: типізована вказівка може бути привласнена вказівці **void**, але не навпаки. Щоб привласнити вказівку **void** типізованій вказівці, необхідно використовувати явне перетворення типів, інакше компілятор видає повідомлення про помилку:

```
void main()
{
  void* vp;
  int* ip;
  int i = 5;
  char* cp;
  char c = 'A';
```

```
  ip = &i; //правильно
  vp = &i; //правильно

  cp = &c; //правильно
  vp = &c; //правильно

  ip = vp; //цього робити неможна
  cp = vp; //цього робити неможна

  vp = ip; //правильно
  vp = cp;  //правильно
}
```

У C++ такі оператори, як **new**, **delete** і **delete[]**, виділяють і звільняють блоки пам'яті в купі. Такий вид пам'яті звичайно називають ***динамічною пам'яттю***, тому що вона створюється і знищується при виконанні програми. Припустимо, необхідно виділити пам'ять для масиву з 10 цілих чисел. Для цього достатньо написати:

```
int* ip;
ip = new int[10];
```

або ще коротше:

```
int* ip = new int[10];
```

Даний код оголошує вказівку **ip** та ініціалізує її значенням, що повернуте **new**. Оператор **new** повертає вказівку на початок виділеного блока пам'яті.

Якщо виділяється пам'ять за допомогою оператора **new**, необхідно зазначити точний її обсяг. Цей обсяг зовсім необов'язково знати під час компіляції, але він повинен бути відомий під час виконання програми:

```
//Введення значення змінної
  int n;
  cin >> n;
//запитується блок пам'яті для збереження n цілих
  int* ip = new int[n];
```

При запиті блоку пам'яті виділяється деякий обсяг пам'яті з глобальної купи. Обсяг розподіленого блоку трохи перевищує запитаний обсяг пам'яті, тому що потрібно ще зберігати додаткову інформацію, щоб адміністратор купи міг управляти блоками.

Оператор **new** поверне ненульову вказівку, навіть якщо запросити нуль байтів. При використанні вказівки, повернутої **new**, неминуче виникнення проблем, тому що для даного блоку пам'яті реально не виділено.

```
void main()
{
//буде повернута значуща вказівка!
```

```
  int* ip = new int[0];
}
```

Оператор **new** дозволяє ініціалізувати зазначеним значенням динамічно виділені дані, як показано нижче:

```
//ініціалізація цілого значенням 10
  int* value = new int(10);
```

Створивши в глобальній купі об’єкт, ви відповідаєте за його видалення. Ця операція для компілятора більш складна, ніж операція створення об’єкта. Чому? Роздивимося такий код:

```
int* ip = new int[10];
delete ip;
```

Скільки пам’яті буде звільнено за допомогою операції **delete**? Оператору **new** передається інформація про число типізованих елементів, для котрих необхідно виділити пам’ять. Оператор **delete** одержує тільки типізовану вказівку. У C++ адміністратор купи використовує вказівку для пошуку виділеного блоку пам’яті. Купа також містить додаткову інформацію, включаючи розмір кожного виділеного блоку. Використовуючи цю приховану інформацію, адміністратор може визначити, скільки звільнити пам’яті.

Для видалення масивів використовується оператор **delete[]**. Такий код показує його використання:

```
char* cp = new char[10];
delete[] cp;
```

Якщо після **delete** не поставити квадратних дужок, то жодного повідомлення про помилку не видається, але помічений як вільний буде тільки перший елемент масиву, а останні опиняться недоступні для подальших операцій. Такі елементи пам'яті називаються сміттям.

Приклад п’ятого засобу завдання початкового значення вказівки. Виведення символу "A".

```
#include <iostream>
using namespace std;

void main()
{
  char* c = (char*)0x003434A0;
  *c = 'A';
  cout<<*c;
}
```

Якщо адреса пам’яті **0x003434A0** буде зайнята будь-яким процесом, в результаті виконання програми з’явиться повідомлення про помилку. Даний засіб ініціалізації вказівок використовується дуже рідко.

Операції з вказівками

З вказівками можна виконувати наступні операції: розадресації, або непряме звернення до об'єкту (*), привласнення, складання з константою, віднімання, інкремент (++), декремент (--), порівняння, приведення типів. При роботі з вказівками часто використовується операція отримання адреси (&). Операція розадресації, або розмикання, призначена для доступу до величини, адреса якої зберігається у вказівці. Цю операцію можна використовувати як для отримання, так і для зміни значення величини (якщо вона не оголошена як константа):

```
    char a; // змінна типу char
    char *p = new char; /* виділення пам'яті під вказівку
та під динамічну змінну типу char */
    *p = 'A'; a = *p; // надання значення обом змінним
```

Як видно з прикладу, конструкцію *ім’я_вказівки можна використовувати в лівій частині оператора привласнення, оскільки вона визначає адресу області пам'яті. Для простоти цю конструкцію можна вважати за ім'я змінної, на яку посилається вказівка. З нею допустимі всі дії, визначені для величин відповідного типу (якщо вказівка проініціалізована). На одну і ту ж область пам'яті може посилатися декілька вказівок різного типу. Застосована до них операція розадресації дасть різні результати. Наприклад, програма

```
#include <iostream>
using namespace std;

void main()
{
 unsigned long int A = 0xcc77ffaa;
 unsigned short int* pint = (unsigned short int*) &A;
 unsigned char* pchar = (unsigned char *) &A;
 printf("| %x | %x | %x |", A, *pint, *pchar);
}
```

В результаті виконання програми на екран виведеться наступний рядок:

```
| cc77ffaa | ffaa | aa |
```

Значення вказівок pint та pchar однакові, але розадресація pchar дає в результаті один молодший байт за цією адресою, а pint – два молодші байти.

У приведеному вище прикладі при ініціалізації вказівок були використані операції приведення типів. Синтаксис операції явного приведення типу простий: перед ім'ям змінної в дужках вказується тип, до якого її потрібно привести. При цьому не гарантується збереження інформації, тому в загальному випадку явних перетворень типу слід уникати. При змішуванні у виразі вказівок різних типів явне перетворення типів потрібне для всіх вказівок, окрім void*. Вказівка може неявно перетворюватися в значення типу bool

(наприклад, у виразі умовного оператора), при цьому ненульова вказівка перетвориться в true, а нульова в false.

Привласнення без явного приведення типів допускається в двох випадках:

– вказівкам типу void*;

– якщо тип вказівок справа та зліва від операції привласнення один і той же.

Таким чином, неявне перетворення виконується тільки до типу void*. Значення 0 неявно перетвориться до вказівки на будь-який тип. Привласнення вказівок на об'єкти вказівкам на функції (і навпаки) неприпустимо. Заборонено і привласнювати значення вказівкам-константам, втім, як і константам будь-якого типу (привласнювати значення вказівкам на константу і змінним, на які посилається вказівка-константа, допускається).

Арифметичні операції з вказівками (складання з константою, віднімання, інкремент та декремент) автоматично враховують розмір типу величин, що адресуються вказівками. Ці операції застосовані тільки до вказівок одного типу і мають сенс в основному при роботі із структурами даних, послідовно розміщеними в пам'яті, наприклад, з масивами. Інкремент переміщує вказівку до наступного елементу масиву, декремент – до попереднього. Фактично значення вказівки змінюється на величину sizeof (тип). Якщо вказівка на певний тип збільшується або зменшується на константу, її значення змінюється на величину цієї константи, помножену на розмір об'єкту даного типу, наприклад:

```
short *p1 = new short [5];
p1++; //значення p збільшується на 2
long *p2 = new long [5];
p2++; //значення q збільшується на 4
```

Різниця двох вказівок – це різниця їх значень, що ділиться на розмір типу в байтах (у застосуванні до масивів різниця вказівок, наприклад, на третій і шостий елементи дорівнює 3). Додавання двох вказівок не допускається. При записі виразів з вказівками слід звертати увагу на пріоритети операцій. Як приклад розглянемо послідовність дій, задану в операторі

```
*p++ = 10;
```

Операції розадресації та інкремента мають однаковий пріоритет і виконуються справа наліво, але, оскільки інкремент постфіксний, він виконується після виконання операції привласнення. Таким чином, спочатку за адресою, записаною у вказівці p, буде записано значення 10, а потім вказівка буде збільшена на кількість байт, відповідно його типу. Те ж саме можна записати таким чином:

```
*p = 10; p++;
```

Вираз (*p)++ навпаки, інкрементує значення, на яке посилається вказівка.

8.2. Посилання

Посилання є синонімом імені (псевдонім), вказаного при ініціалізації посилання. Посилання можна розглядати як вказівку, яка завжди розіменовується. Формат оголошення посилання:

тип & ім'я;

де тип – це тип величини, на яку вказує посилання; & – оператор посилання, який означає, що наступне за ним ім'я є ім'ям змінної типу посилання, наприклад:

```
int x;
int& t = x; // посилання t – альтернативне ім'я для x (псевдонім)
const char& cc = '\n'; // посилання на константу
```

Треба запам'ятати наступні правила:

1. Змінна-посилання повинна явно ініціалізуватися при її описі, крім випадків, коли вона є параметром функції.

2. Після ініціалізації посиланню не може бути привласнена інша змінна.

3. Тип посилання повинен збігатися з типом величини, на яку воно посилається.

4. Не дозволяється визначати вказівки на посилання, створювати масиви посилань та посилання на посилання.

Посилання застосовуються найчастіше як параметри функцій і типів повертаємих функціями значень. Посилання дозволяють використовувати у функціях змінні, які передаються за адресою, без операції розадресації, що покращує читаємість програми. Посилання, на відміну від вказівки, не займає додаткового простору в пам'яті і є просто іншим ім'ям величини. Операція над посиланням приводить до зміни величини, на яку вона посилається.

Посилання – це псевдоніми для об’єктів.

Для встановлення псевдоніму, він повинен бути того ж типу, що і змінна.

```
int i;
int &ir = i;
```

Для змінної **i** встановлюється псевдонім **ir**. Присвоювання **ir** значення та його використання значення дасть той же результат, що і присвоювання значення та використання **i**.

```
ir = 3;
j = ir * i;     //результат 9
```

8.3. Одновимірні масиви

При використанні простих змінних кожної області пам'яті для зберігання даних використовується своє ім'я. Якщо з групою величин однакового типу потрібно виконувати одноманітні дії, їм дають одне ім'я, а розрізняють по порядковому номеру. Це дозволяє компактно записувати безліч операцій за допомогою циклів. Кінцева іменована послідовність однотипних величин називається масивом. Опис масиву в програмі відрізняється від опису простої змінної наявністю після назви квадратних дужок, в яких задається кількість елементів масиву (розмірність):

```
float a [10]; // опис масиву з 10 дійсних чисел
```

Елементи масиву нумеруються з нуля. При описі масиву використовуються ті ж типи даних, що і для простих змінних. Значення, якими треба ініціалізувати масив при оголошенні, записуються у фігурних дужках. Значення елементам привласнюються по порядку. Якщо елементів в масиві більше, ніж проініціалізованих, елементи, для яких значення не вказані, заповнюються нулями:

```
int b[5]= {3, 2, 1};
// b[0] = 3; b[l] = 2; b[2] = l; b[3] = 0; b[4] = 0;
```

Розмірність масиву разом з типом його елементів визначає об'єм пам'яті, необхідний для розміщення масиву, це виконується на етапі компіляції, тому розмірність може бути задана тільки цілою додатною константою або константним виразом. Якщо при описі масиву не вказана розмірність, компілятор виділить пам'ять по кількості значень, що були проініціалізовані при оголошенні масиву.

Щоб отримати доступ до елементу масиву після його імені вказується номер елементу (індекс) в квадратних дужках.

Приклади оголошення та ініціалізації одновимірних масивів:

Оголошення масивів.

```
int a[10],ss[20];
double cc[15],k[4];
```

Ініціалізація під час оголошення.

```
int m[5]={2,3,4,5,6};
```

або

```
int m[]={2,3,4,5,6};
double s[3]={1.23,4.57,8.675};
```

Для ініціалізації всіх елементів великого масиву нулями під час оголошення, проініціалізуйте його перший елемент, інші автоматично заповняться нулями.

Якщо масив оголошено глобально, він автоматично заповнюється нулями.

Ідентифікатор масиву є константною вказівкою на його нульовий елемент. Для масиву з ім'ям **x** – це те ж саме, що **&x[0]**, а до **i**-го елементу масиву можна звернутися, використовуючи вираз ***(x+i)**. Можна описати вказівку, привласнити їй адресу початку масиву і працювати з масивом через вказівку. Наступний фрагмент програми копіює всі елементи масиву **x** в масив **y**:

```
int x[10] = {9,8,7,6,5,4,3,2,1,0};
int y[10];
int *px = x; // або int *px = &x[0];
int *py = y;
for(int i = 0; i<10; i++)
*py++ = *px++; //або py[i]= px[i];
```

Розглянемо приклади роботи з одновимірними масивами.

Приклад 1. Знаходження в масиві максимального значення.

```
#include <iostream>
using namespace std;

#define SIZE 10

void main()
{
//Ініціалізація під час оголошення
  int ara[SIZE]={5,2,7,9,34,56,6,4,285,17};
  int i,max;
//Виведення масиву на екран
  for (i = 0; i < SIZE; i++)
    cout << ara[i] <<"\n";
  //cout << *(ara+i) <<"\n";
  max=ara[0];
  for(i = 1; i < SIZE; i++)
    if (ara[i] > max) max = ara[i];
  cout << "max = "<<max;
}
```

Приклад 2. Знаходження максимального (max) та мінімального (min) значення, заповнюючи масив довільними значеннями від 0-99.

```
#include <iostream>
#include <ctime>
using namespace std;

#define SIZE 10
```

```
void main()
{
  int ara[SIZE];
  int i, max, min;
//Включення генератору псевдовипадкових чисел
  srand(time(0));

  for (i = 0;i < SIZE;i++)
    ara[i] = rand()%100;
//Виведення масиву на екран
  for (i = 0; i < SIZE; i++)
    cout << ara[i] <<"\n";
  //cout << *(ara+i) <<"\n";
  max = min = ara[0];
  for (i = 1; i < SIZE; i++)
  {
    if (ara[i] > max) max = ara[i];
    if (ara[i] < min) min = ara[i];
  }
  cout << "min = "<< min <<"\n";
  cout << "max = "<<max;
}
```

Приклад 3. Пошук числа в масиві, яке дорівнює введеному значенню. Якщо такого немає, то воно додається в масив. Передбачається, що число введених значень, що не повторюються, не перевищує 100.

```
#include <iostream>
using namespace std;

#define MAX 100

void main()
{
  int i,num = 0;
  int ss,pt[MAX];
  do
  {
    cout << "Введи число ->";
    cin >> ss;
    if (ss == -1) break; //Вихід за бажанням
                         //користувача
    for (i = 0; i < num; i++)
      if (ss == pt[i]) break;
    if (i == num)
    {
```

```
      pt[num] = ss;
    //*(pt+num) = ss;
      num++;
    }
  }
  while (num < MAX);
  cout << "Результат: \n";
  for (i = 0; i < num; i++)
    cout << pt[i] << "\n";
  //cout << *(pt+i) << "\n";
}
```

Розміщення одновимірного масиву в пам'яті.

Для C++ ім'я масиву є дійсною адресою, по якій в пам'яті знаходиться перший елемент масиву. Припустимо, що ви визначаєте масив з ім'ям **amo**:

```
int amo[6]={4,1,3,7,9,2};
```

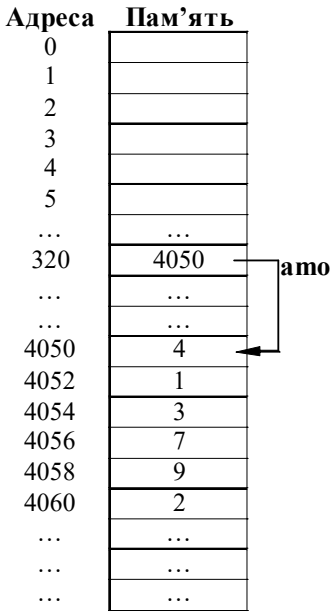


Рис.8.1. Розміщення масиву **amo** у пам'яті.

На рис. 8.1. показано, як масив розміщується в пам'яті.

Відповідно до рисунку масив починається з адреси 4050, реальна адреса змінних встановлюється під час завантаження скомпільованої програми. Можна звертатися до масиву за допомогою звичайних правил з індексами або за допомогою зміни адреси масиву.

Записи **amo[3]**, **(amo+3)[0]** та ***(amo+3)** ідентичні.

У кожному з таких рядків відбувається звертання до 4-го елемента масиву.

(amo+0)[3] ***(amo+0+3)**
(amo+2)[1] ***(amo+2+1)**
(amo-2)[5] ***(amo-2+5)**
(1+amo)[2] ***(1+amo+2)**

Наведемо приклад виведення окремих елементів масиву, використовуючи вказівки.

```
#include <iostream>
#include <iomanip>
using namespace std;

void main()
{
  float ara[ ]={100,200,300,400,500};
  float *fp = &ara[0];
  cout << setiosflags(ios::fixed)
       << setprecision(1);
  cout << *fp << "\n";
  fp++;
  cout << *fp << "\n";
  fp++;
  cout << *fp << "\n";
  fp=ara;
  cout << *(fp+2) << "\n";
  cout <<(fp+0)[0] << " "<< (ara+0)[0] << "\n";
  cout << (fp+4)[0] <<" "<< (ara+4)[0] << "\n";
  cout << (fp-1)[2] <<" "<< (ara-1)[2] << "\n";
}
```

Результат:

```
100.0
200.0
300.0
300.0
100.0 100.0
500.0 500.0
200.0 200.0
```

При роботі з одновимірними масивами слід звернути увагу на наступні особливості:

– Незважаючи на те, що C++ автоматично не заносить у пам'ять нулі (або які-небудь інші значення), якщо під час оголошення ви проініціалізуєте декілька елементів, інші заповняться нулями.

– На відміну від інших мов, C++ дозволяє привласнити значення незарезервованим членам масиву. Але це не варто робити, тому що ви зіпсуєте

інші дані або код.

Так не можна ініціалізувати масив: ~~`int mas[];`~~

У цьому випадку все правильно: `int mas[]={1,2,3};`

– Використання визначальних констант (#define) має переваги. При недостачі елементів у масиві можна змінити тільки один рядок у #define.

Динамічні одновимірні масиви.

Динамічні масиви створюють за допомогою оператору **new**, при цьому необхідно вказати тип та розмірність, наприклад:

```
int n = 100;
float *p = new float [n];
```

У цьому рядку створюється змінна-вказівка на **float**, в динамічній пам'яті відводиться безперервна область, достатня для розміщення 100 елементів дійсного типу, і адреса її початку записується у вказівку **p**. Динамічні масиви не можна ініціалізувати при створенні, а також вони не обнуляються.

Перевага динамічних масивів полягає в тому, що розмірність може бути змінною, тобто об'єм пам'яті, що виділяється під масив, визначається на етапі виконання програми. Доступ до елементів динамічного масиву здійснюється таким же чином, як і до статичних, наприклад, до елементу номер 5 приведеного вище масиву можна звернутися як p[5] або *(p+5).

Пам'ять, зарезервована під динамічний масив за допомогою **new []**, повинна звільнятися оператором **delete []**, наприклад:

```
delete []p;
```

При невідповідності способів виділення і звільнення пам'яті результат не визначений. Розмірність масиву в операторі **delete** не вказується, але квадратні дужки обов'язкові.

Розглянемо приклад знаходження максимального та мінімального елементів масиву, використовуючи оператори **new** та **delete[]**.

```
#include <iostream>
#include <ctime>
using namespace std;

void main()
{
  int i, n, max, min;
//Включення генератору псевдовипадкових чисел
  srand(time(0));
  cout <<"n = ";
  cin>>n;
  int *ara = new int[n];

  for (i = 0;i < n;i++)
```

```
    ara[i] = rand()%100;
//Виведення масиву на екран
  for (i = 0; i < n; i++)
    cout << ara[i] <<"\n";
  //cout << *(ara+i) <<"\n";
  max = min = ara[0];
  for (i = 1; i < n; i++)
  {
    if (ara[i] > max) max = ara[i];
    if (ara[i] < min) min = ara[i];
  }
  cout << "min = "<< min <<"\n";
  cout << "max = "<<max;

delete[] ara;
}
```

8.4. Багатовимірні масиви

Багатовимірні масиви задаються окремими вимірами в квадратних дужках, наприклад, оператор `int matr [3][4];` задає опис двовимірного масиву з 3 рядків та 4 стовпців. У пам'яті такий масив розташовується послідовно по рядках. Тобто в пам’яті будь-який багатовимірний масив розташовується як одновимірний. Двовимірний масив представляє собою константну вказівку, яка збільшується на кількість елементів в кожному рядку масиву. Привласнимо двовимірний масив вказівці.

```
int matr [3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
int (*mm)[4] = matr;
```

Тепер перший та другий рядок повністю еквівалентні, з тією відмінністю, що **matr** – константна вказівка, а **mm** – ні.

Запис **int (*mm)[4]** означає, що при арифметичних операціях з вказівкою, вона збільшується або зменшується на кількість байтів 4-х елементів типу даних **int** до першого розмикання. Після першого розмикання адреса збільшується на кількість байтів одного елементу **int**, як у разі звичайної вказівки.

Для доступу до елементу багатовимірного масиву вказують всі його індекси, наприклад, `matr[i][j]`, або іншим способом: `*(matr[i]+j)` або `*(*(matr+i)+j)`.

При ініціалізації багатовимірного масиву він представляється або як масив з масивів, при цьому кожен масив береться в свої фігурні дужки (в цьому випадку ліву розмірність при описі можна не вказувати), або задається загальний список елементів в тому порядку, в якому елементи розташовуються в пам'яті:

I-й варіант	II-й варіант

int aaa[3][2]=
{
{5,4},
{7,1},
{3,0}
};
	int aaa[3][2]=
	{5,4,7,1,3,0};

Як і у випадку роботи з одновимірним масивом, якщо ви ініціалізуєте один або більше елементів, інші – заповнюються нулями.

Приклад:

```
float ss[3][2]={0.0};
```

всі елементи масиву заповнюються нулями.

Наведемо приклад виведення певного елементу двовимірного масиву різними засобами з використанням вказівок.

```
#include <iostream>
using namespace std;

void main()
{
  int ara[][2]={{100,200},{300,400},{500,600}};
  int (*mm1)[2] = ara;
  int *mm2 = (int*)ara;

  cout << ara[1][1] <<"\t" <<*(*(ara+1)+1)<<"\n";
  cout << mm1[1][1] <<"\t" <<*(*(mm1+1)+1)<<"\n";
  cout << (*ara)[3] <<"\t" <<*(*ara+3)<<"\n";
  cout << mm2[3] <<"\t" <<*(mm2+3)<<"\n";
}
```

Результат:

```
400     400
400     400
400     400
400     400
```

Змінні **int ara[][2]** та **int (*mm1)[2]** представляють собою вказівки, де запис у квадратних дужках показує на скільки елементів треба зміщуватись при арифметичних операціях. У нашому випадку це **2*sizeof(int)**. Після першого розмикання вказівка буде переміщуватись через 4 байти **(sizeof(int))**.

За допомогою рядку коду `int *mm2 = (int*)ara;` явно приводимо вказівку **ara** до типу **(int*)** після чого з двовимірним масивом можна працювати як з одновимірним.

Оператор **for** є гарним засобом організації доступу до кожного елемента двовимірної таблиці

```
for (row=0;row<2;row++)
for (col=0,col<3;col++)
cout <<row<" "<<col<<"\u";
```

Наведемо приклад виведення цін на дискети:

Комп'ютерна компанія продає дискети 3.5 і 5.25 дюйма. Кожна дискета буває однієї з 4-х ємностей: односторонніми, подвійної щільності, двосторонніми подвійної щільності, односторонніми підвищеної щільності, двосторонніми підвищеної щільності.

```
#include <iostream>
#include <iomanip>
using namespace std;

void main()
{
  int row, col;
  float disk[2][4] =
          {{2.3f, 2.75f, 3.2f, 3.5f},
           {1.75f, 2.1f, 2.6f, 2.95f}};
  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout << "\tSingle sided,\tDouble sided,"
       << "\tSingle sided,\tDouble sided,\n";
  cout << "\tDouble density\tDouble density"
       << "\tHigh density \tHigh density \n";
  for (row = 0; row < 2; row++)
  {
    if(row == 0)
      cout << "3.5\" \t";
    else
      cout << "5.25\"\t";
    for (col = 0; col < 4; col++)
      cout << "$" << setprecision(2)
           << disk[row][col] << "\t\t";
    cout << "\n";
  }
}
```

Результат:

```
          Single  Double    Single   Double
          sided,  sided,    sided,   sided,
          Double  Double    High     High
          density density   density  density
    3.5   $2.30   $2.75     $3.20    $3.50
    5.25  $1.75   $2.10     $2.60    $2.95
```

Наступний приклад представляє найбільший інтерес. У ньому продемонстровано як за рахунок приведення вказівок можна перетворити двовимірний масив в одновимірний і навпаки – одновимірний в двовимірний.

```
#include <iostream>
using namespace std;
void main()
{

 int i,j;

//З 2-мірного в 1-й

 int mas[3][3]=
                          {
                    {11,12,13},
                    {21,22,23},
                    {31,32,33}
                           };

int* aa = (int*)mas;

for(i=0;i<9;i++)
{
 cout<<aa[i]<<"   ";
}

cout<<"\n\n\n";

//З 1-мірного в 2-й
int a[9]= {1, 2, 3, 4, 5, 6, 7, 8, 9};
int (*b)[3];

b = (int(*)[3])a;

for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
     cout<<b[i][j]<<"   ";
 cout<<"\n";
}

}
```

Перетворення двовимірного масиву **mas[3][3]** в одновимірний здійснюється за допомогою рядка коду:

```
int* aa = (int*)mas;
```

Тобто вказівка **int(*)[3]** приводиться до вказівки **int***.

Перетворення одновимірного масиву **a[9]** в двовимірний здійснюється за допомогою рядка коду:

```
b = (int(*)[3])a;
```

Тобто вказівка **int*** приводиться до вказівки **int(*)[3]**. При цьому вказівка **b** оголошується таким чином:

```
int (*b)[3];
```

Після виконання програми отримаємо наступний результат:

```
11 12 13 21 22 23 31 32 33
1 2 3
4 5 6
7 8 9
```

Знаючи особливості приведення вказівок при роботі з масивами можна маніпулювати їх вимірюваннями як завгодно. Тобто, перетворити масив з одного вимірювання в інше не представляє ніякої складності, оскільки в пам'яті всі елементи статичних масивів розташовуються послідовно без виділення додаткової пам'яті.

Можна працювати з багатовимірними масивами, виділяючи їм динамічну пам’ять з використанням оператору **new**. Для кожного вимірювання необхідно виділити додаткову пам’ять.

Приведемо приклад.

```
#include <iostream>
using namespace std;

void main()
{
 //m, n - відповідно кількість рядків
 // і стовпців масиву

 int i, j, m, n;
 cin>>m>>n;

//Ініціалізація масиву
 int **mas =   new int*[m];
 for (i=0;i<m;i++)
        mas[i]= new int[n];

 //Введення даних
 for(i=0;i<m;i++)
     for(j=0;j<n;j++)
```

```
    cin>>mas[i][j];

 //Виведення даних
 for(i=0;i<m;i++)
  {
     for(j=0;j<n;j++)
       cout<<mas[i][j]<<" ";
          cout<<"\n";
   }
 //Видалення масиву
 for (i=0;i<m;i++)
    delete[] mas[i];

 delete [] mas;
}
```

8.5. Символьні масиви

Рядкових змінних у C++ немає, для цього використовуються масиви символів (символьні масиви) Символом кінця рядку є символ ASCII-нуля, '\0'.

Припустимо необхідно зберегти ім'я людини, вік, зарплату. Числові змінні віку та зарплати оголошуються наступним чином:

```
int age;
float salary;
```

Рядкова змінна не має місця, але можна створити масив символів:

```
char name[15];
```

У пам'яті резервується місце під масив, який складається з 15 символів.

Під час оголошення масиву символів йому можна привласнити значення рядку.

```
char name[6] = "Маша";
```

У пам'яті ЕОМ:

[0]	М
[1]	А
[2]	Ш
[3]	А
[4]	\0

Можна одержати доступ до визначеного елемента масиву:

```
name[0] = 'П';
```

У пам'яті ЕОМ:

[0]	П
[1]	А
[2]	Ш
[3]	А
[4]	\0

При виведенні результату можна зазначити тільки ім’я рядку:

```
cout<<name;          //Результат: Паша
```

Можна оголосити масив символів наступним чином:

```
char cc[] = {'a', 'b', 'c'};
```

Цей масив не може бути рядком, просто він містить три різних символи.

```
char bb = cc[1];     // Змінній bb привласнено
                     // значення 'b'
cout<<cc;
```

<u>ПОМИЛКА!</u> Наприкінці ланцюжка символів немає ASCII-нуля, виведення відбуватиметься доти, поки не зустрінеться '\0'.
Програма помилкова.

```
void main()
{
char name[20];
name = "Петя";    //Не можна імені масиву
                  //привласнити значення, що
                  //можливо тільки при
                  //оголошенні.
}
```

Для ініціалізації масиву існує **5 основних засобів**. 3 з них ініціалізуються під час оголошення.

```
#include <iostream>
using namespace std;

void main()
{
// 1-й засіб
char f1[] = "Привіт";

// 2-й засіб
char f2[7] = "Привіт";

// 3-й засіб
char f3[] = {'П','р','и','в','і','т','\0'};
```

```
// 4-й засіб
char f4[7];
//використання бібліотечної функції
strcpy(f4, "Привіт");

// 5-й засіб
char f5[7];
f5[0]= 'П';
f5[1]= 'р';
f5[2]= 'и';
f5[3]= 'в';
f5[4]= 'і';
f5[5]= 'т';
f5[6]= '\0';
}
```

При виведенні рядків із символьних масивів робота з їхніми елементами шляхом зміни адрес більш корисна, ніж процедура з масивами цілих чисел.

Приклад:

```
char nam[]={'T','e','d','\0','E','v','a','\0',
'S','a','m','\0'};
```

Номер елемента	Значення
[0]	T
[1]	E
[2]	D
[3]	\0
[4]	E
[5]	V
[6]	A
[7]	\0
[8]	S
[9]	A
[10]	M
[11]	\0

```
cout <<nam;          //Надрукує Ted
cout <<nam+4;        //Надрукує Eva
cout <<nam+8;        //Надрукує Sam
```

Використання вказівок на символи.

Розглянемо 2 рядки:

```
char cara[]="Привіт";
char *cp = "Москва";
```

Після початкової ініціалізації у вказівці на символьну змінну міститься адреса першого символу рядка.

```
cara = "Привіт";  //Невірно
ср = "Київ";      //Можна: занесення у вказівку
                  //нового рядка.
```

Приклад:

```
#include <iostream>
using namespace std;

void main()
{
  char name[20] = "Іван Петров";
  char *t = name;
  strcpy(t+5,"Сидоров");
  cout <<t; //Іван Сидоров
}
```

Запам’ятовування масивів рядків.

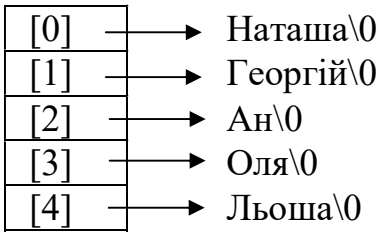
```
char names[5][20]=
{
  {"Наташа"},
  {"Георгій"},
  {"Ан"},
  {"Оля"},
  {"Льоша"}
};
```

Таблиця займає в пам’яті багато місця. Нераціонально використовується пам’ять.

Для усунення проблем раціонального використання масиву символьних вказівок, кожна вказівка містить адресу рядка в пам’яті, і ці рядки можуть мати різну довжину.

1-й варіант	2-й варіант
`char *names[5]= { {"Наташа"}, {"Георгій"}, {"Ан"}, {"Оля"}, {"Льоша"} };`	`char *names[]= { "Наташа", "Георгій", "Ан", "Оля", "Льоша" };`

Рядки можуть знаходитися в будь-якому масиві пам’яті




```
cout << *names;                 //Друкує Наташа
cout << *(names+1);             //Георгій
```

Робота з вказівками на рядки набагато ефективніше, ніж з самими рядками. Наприклад, сортування рядків за допомогою масиву вказівок виконується швидше. Під час сортування змінюються тільки вказівки, а не вміст рядків.

Приклад:

При запровадженні чисел від 1 до 7 на екран виводиться день тижня

```
#include <iostream>
using namespace std;

void main()
{
  char *dn[]=
  {
    "Понеділок",
    "Вівторок",
    "Середа",
    "Четвер",
    "П’ятниця",
    "Субота",
    "Неділя"
  };
  int day_num;
  do
  {
    cout <<"Введіть день тижня(1-7) ";
    cin >> day_num;
  }
  while ((day_num < 1)||(day_num > 7));
  cout << *(day_num - 1 + dn) << "\n";
}
```

Контрольні питання

1. Що таке вказівка?
2. Як оголосити вказівку та взяти адресу змінної?
3. Призначення вказівки void *.
4. Як привести вказівку до потрібного типу даних?
5. Призначення операторів new, delete та delete [].
6. Що таке посилання?
7. Призначення та використання посилань.
8. Що таке масив?
9. Які існують засоби ініціалізації одновимірного масиву?
10. Організація введення/виведення масиву.
11. Наведіть приклад введення/виведення одновимірного масиву.
12. Як розміщується одновимірний масив в пам’яті?
13. Що таке багатовимірний масив?
14. Як знайти мінімальний та максимальний елементи масиву?
15. Як вказівка пов’язана з одновимірним масивом? Використання арифметичних операцій з вказівками.
16. Яким чином перетворити двовимірний масив до одновимірного? Наведіть приклад.
17. Яким чином перетворити одновимірний масив до двовимірного? Наведіть приклад.

Завдання

Одновимірні масиви

У даній роботі необхідно зробити програму роботи з одновимірними масивами. Бажано використовувати динамічні масиви (виділення пам’яті за допомогою оператора new). Введення даних організовувати, використовуючи генерацію випадкових чисел.

Приклад виконання завдання.

Ввести масив чисел, знайти у ньому максимальне значення, значення елементів, які є парними числами, збільшити на абсолютне значення максимального елементу.

Код програми має наступний вигляд:

```
#include <iostream>
#include <ctime>
#include <cmath>
using namespace std;

void main()
{
```

```
 int i,n,max;
 int *mas;

 cout<<"Введите количество элементов массива n ";
 cin>>n;

 cout<<"\n\n";

// Виділення динамічної пам’яті
mas = new int[n];

// Генерація випадкових чисел
srand(time(0));

//введення даних
 for(i = 0;i<n;i++)
     mas[i] = rand()%101 - 50;
 //вывод данных
 for(i = 0;i<n;i++)
     cout<<mas[i]<<"\t";

//розрахунок
 max = mas[0];
 for(i = 1;i<n;i++)
     if(mas[i]>max)max = mas[i];

 for(i = 0;i<n;i++)
     if(!(mas[i]&1))mas[i]+=abs(max);

 cout<<"max = "<<max<<"\n\n";

 //виведення даних
 for(i = 0;i<n;i++)
     cout<<mas[i]<<"\t";

// Видалення масиву
delete []mas;
}
```

Варіанти індивідуальних завдань

1. Визначити добуток додатних парних чисел масиву B(n). Якщо таких елементів немає, вивести на екран дисплея повідомлення: «Додатних парних елементів в масиві немає».
2. Визначити добуток від’ємних чисел масиву A(n), що стоять на парних позиціях. Якщо таких елементів немає, вивести на екран дисплея повідомлення:

«Від’ємних елементів в масиві немає».

3. Визначити суму і кількість непарних чисел масиву M(n). Якщо таких елементів немає, вивести на екран дисплея повідомлення: «Непарних елементів в масиві немає».

4. У масиві чисел K(n) визначити суму і кількість чисел, кратних 5. Якщо таких елементів немає, вивести на екран дисплея повідомлення: «Елементів кратних 5 в масиві немає».

5. У масиві чисел X(n) визначити кількість чисел, кратних 2, і чисел, не кратних 3. Якщо таких елементів немає, вивести на екран дисплея повідомлення: «Чисел, кратних 2 і чисел не кратних 3 в масиві немає».

6. Є масив чисел C(n), визначити кількість і добуток елементів масиву, які знаходяться в діапазоні $0 \le C(i) \le 7$. Якщо таких елементів немає, вивести на екран дисплея повідомлення: «Елементів з діапазону [0; 7] в масиві немає».

7. У заданому масиві A(n) визначити середнє арифметичне значення чисел, кратних трьом. Якщо таких елементів немає, вивести на екран дисплея повідомлення: «Елементів кратних 3 в масиві немає».

8. Визначити середнє арифметичне значення елементів масиву F(n), які задовольняють вимозі $-3 \le F(i) \le 5$. Якщо таких елементів немає, вивести на екран дисплея повідомлення: «Елементів, що задовольняють вимозі, в масиві немає».

9. У числовому масиві A(n) визначити мінімальний елемент масиву і його порядковий номер. Результат вивести на екран дисплея.

10. Визначити мінімальний елемент числового масиву K(n) і кількість елементів, рівних мінімальному елементу. Вивести на екран дисплея результат.

11. Визначити добуток непарних елементів масиву P(n), що стоять на парних позиціях. Якщо таких елементів немає, на екран дисплея вивести повідомлення: «Непарних елементів, що стоять на парних позиціях в масиві немає».

12. Визначити середнє арифметичне значення елементів масиву B(n), кратних восьми. Якщо таких елементів немає, на екран дисплея вивести повідомлення: «Елементів кратних 8 в масиві немає».

13. У числовому масиві Z(n) серед додатних елементів визначити максимальний елемент масиву. Вивести на екран дисплея результат.

14. Визначити кількість від’ємних елементів в масиві A(n) і на їх місце записати нулі. Якщо таких елементів немає, на екран дисплея вивести повідомлення: «Від’ємних елементів в масиві немає».

15. Визначити добуток додатних елементів масиву і їх кількість. За відсутності додатних чисел вивести на екран дисплея «Додатних чисел в масиві немає».

16. Обчислити кількість елементів цілочисельного масиву V(n), кратних семи. За відсутності таких елементів вивести на екран дисплея «Елементів, кратних 7, немає».

17. Для числового масиву Z(n) визначити середнє арифметичне значення мінімального і максимального елементів. Результат вивести на екран дисплея.

18. Для заданого масиву A(n) обчислити суму і кількість елементів, що

задовольняють умові $2 \le A(i) \le 10$. Результат вивести на екран дисплея.

19. Визначити суму і добуток додатних чисел, що стоять на парних позиціях в масиві B(n). Результат вивести на екран дисплея.

20. У масиві C(n) визначити середнє арифметичне значення додатних елементів і середнє арифметичне значення від'ємних елементів. Результат вивести на екран дисплея.

21. У масиві B(n) визначити окремо кількість від'ємних чисел, кількість додатних чисел і чисел, рівних нулю. Результат вивести на екран дисплея.

22. Визначити суму додатних парних чисел масиву B(n). У разі відсутності додатних парних чисел вивести на екран дисплея «Додатних парних чисел в масиві немає».

23. Визначити середнє арифметичне значення чисел масиву A(n), що стоять на парних позиціях, і середнє арифметичне значення чисел масиву A(n), що стоять на непарних позиціях. Результат вивести на екран дисплея.

24. Визначити суму додатних елементів масиву B(n), що стоять на позиціях, кратних трьом. Результат вивести на екран дисплея.

25. Заданий масив цілих чисел C(n). Обчислити добуток максимального і мінімального елементів. Результат вивести на екран дисплея.

26. У масиві чисел B(n) всі по додатні елементи замінити на нульові значення і визначити середнє арифметичне значення від'ємних елементів масиву. Результат вивести на екран дисплея.

27. У масиві чисел C(n) числа, що стоять на парних позиціях, замінити на нулі і визначити середнє значення чисел, що стоять на непарних позиціях. Вивести на екран дисплея результат обчислення.

28. У числовому масиві A(n) визначити індекси мінімального і максимального елементів. Визначити різницю між максимальним та мінімальним елементами. Результат вивести на екран дисплея.

29. Для заданого цілочисельного масиву M(n) обчислити добуток і кількість від'ємних непарних чисел, розташованих на парних позиціях. Результат вивести на екран дисплея.

30. Заданий цілочисельний масив A(n). Обчислити суму і кількість чисел, що діляться без остачі на 5. Якщо таких чисел немає, на екран дисплея необхідно видати повідомлення «Чисел, що діляться без остачі на 5, в масиві немає».

31. У заданому масиві A(n), розділити всі елементи масиву A(n) на п'ятий елемент масиву A(n). У перетвореному масиві обчислити суму елементів, що стоять на непарних позиціях. Результат вивести на екран дисплея.

32. Заданий масив чисел X(n). Обчислити кількість від'ємних чисел і вивести на екран дисплея індекс першого від'ємного елементу і індекс останнього від'ємного елементу.

33. Є цілочисельний масив B(n). Перетворити заданий масив шляхом множення кожного елементу масиву на максимальний елемент цього масиву. Вивести на екран дисплея перетворений масив і максимальний елемент цього масиву.

34. Заданий масив чисел A(n). Перетворити заданий масив шляхом ділення кожного парного елементу масиву на 2. Обчислити середнє арифметичне значення елементів перетвореного масиву. Вивести на екран дисплея. перетворений масив і отриманий результат.

35. У масиві C(n) визначити кількість елементів рівних нулю і на їх місце записати число 5. Обчислити середнє арифметичне значення елементів, рівних 5. Вивести на екран дисплея перетворений масив і результат обчислення.

Двовимірні масиви

У даній роботі необхідно зробити програму роботи з двовимірними масивами. Введення даних організовувати, використовуючи генерацію випадкових чисел.

Приклад виконання завдання.
Знайти суму елементів головної та побічної діагоналі квадратної матриці.
Код програми має наступний вигляд:

```
#include <iostream>
#include <ctime>
using namespace std;

void main()
{
 int i,j,n;
 int mas[30][30];
 double s1,s2;

 cout<<"Введите размерность матрицы n ";
 cin>>n;

 cout<<"\n\n";

//генерація випадкових чисел
srand(time(0));

//введення даних
 for(i = 0;i<n;i++)
     for(j = 0;j<n;j++)
         mas[i][j] = rand()%101 - 50;

 //виведення даних
 for(i = 0;i<n;i++)
 {
     for(j = 0;j<n;j++)
         cout<<mas[i][j]<<"\t";
```

```
 cout<<"\n";
}

//розрахунок
s1 = s2 = 0;
for(i = 0;i<n;i++)
    {
   s1 += mas[i][i];
   s2 += mas[i][n-i-1];
  }

 cout<<"\ns1 = "<<s1;
 cout<<"\ns2 = "<<s2;
}
```

Варіанти індивідуальних завдань

1. Для кожного рядка заданої матриці A(n, m) знайти номера стовпців, які містять ненульові елементи
2. Для кожного рядка заданої матриці A(n, m) знайти номера стовпців, які вмішують від'ємні елементи.
3. Для кожного стовпця заданої матриці A(n, m) знайти номери рядків, які вмішують додатні елементи.
4. Для кожного рядка заданої матриці A(n, m) визначити мінімальний елемент та номер стовпця, в якому він знаходиться.
5. Для заданої патриці A(n, m) визначити середнє арифметичне усіх додатних елементів. У випадку їх відсутності вивести повідомлення “Додатних чисел у матриці немає”.
6. Для кожного рядка заданої матриці A(n, m) визначити суму, серед одержаних значень визначити максимальне.
7. Для кожного стовпця заданої матриці A(n, m) знайти добутки, серед яких знайти найменше значення.
8. Для заданої матриці A(n, m) визначити середнє арифметичне значення для кожного стовпця.
9. Сформувати матрицю, кожний елемент якої уявляє собою дробову частину відповідного елемента заданої матриці A(n, m).
10. В заданій матриці A(n, m) упорядкувати елементи кожного рядка по зростанню.
11. Сформувати матрицю, кожний елемент якої уявляє собою частку від ділення цього елементу на перший елемент відповідного рядка.
12. Для кожного стовпця заданої матриці A(n, m) визначити максимальний елемент і номер рядка, в якому він знаходиться.
13. Помножити кожний елемент стовпця матриці A(n, m) на перший елемент даного стовпця.
14. Визначити мінімальний та максимальний елементи матриці A(n, m), а також їх координати.

15. Для заданої матриці A(n, m) визначити середнє арифметичне значення для кожного рядка.

16. Визначити кількість від'ємних елементів в кожному стовпці заданої матриці A(n, m).

17. Для заданої матриці A(n, m) визначити середнє арифметичне усіх від'ємних елементів. У випадку їх відсутності вивести повідомлення “Від'ємних чисел у матриці немає”.

18. Розмістити елементи кожного рядка матриці A(n, m) у порядку зменшення значень.

19. Із заданої матриці A(n, m) сформувати і надрукувати матрицю, в якій елементи, що задовольняють умові $-1 < A(i, j) < 1$, замінити нулями.

20. Розмістити елементи кожного стовпця матриці A(n, m) у порядку зменшення значень.

21. Визначити середнє значення додатних елементів, які розмішені на головній діагоналі квадратної матриці A(n, n). У випадку їх відсутності вивести повідомлення " Додатних елементів на головній діагоналі матриці немає".

22. Сформувати матрицю, кожний елемент якої є часткою від ділення цього елементу на останній елемент відповідного стовпця.

23. Розмістити елементи головної діагоналі квадратної матриці A(n, n) у порядку зменшення значень.

24. Із заданої квадратної матриці A(n, n) сформувати матрицю, в якій елементи, що знаходяться не на головній діагоналі, замінити нулями.

25. Із заданої матриці A(n, m) сформувати матрицю, в якій елементи, що знаходяться на непарних позиціях, замінити нулями.

Варіанти індивідуальних завдань (ускладнений варіант)

1. У заданій квадратній матриці A(n, n) знайти суму недіагональних елементів. Серед цих елементів знайти максимальний і мінімальний елементи та їх координати.

2. Дано квадратну матрицю A(n, n). Скласти новий масив B(n), в якому кожний елемент є першим додатнім елементом кожного рядка. Якщо таких елементів немає, тоді B(i)=-1.

3. Дано квадратну матрицю A(n, n). Скласти новий масив B(n), в якому кожний елемент є найбільшим елементом відповідного стовпця матриці.

4. У матриці A(n, m) визначити суму елементів рядків, стовпців і діагоналей. Серед сум визначити найменшу.

5. У матриці A(n, m) знайти максимальний від'ємний елемент із парними індексами. Визначити кількість від'ємних елементів і їх координати.

6. Задана матриця A(n, m). Знайти номер першого стовпця, який має максимальну кількість від'ємних чисел.

7. Задана квадратна матриця A(n, n). Визначити суму елементів, розмішених нижче головної діагоналі. Серед цих елементів визначити

максимальний додатній елемент та його координати. Якщо таких елементів немає, вивести повідомлення : " Додатних елементів немає ".

8. Задана матриця A(n, m). Знайти номер рядка, який вміщує мінімальну кількість нульових елементів.

9. Задана матриця A(n, m). Скласти нову матрицю, кожний елемент якої є часткою від ділення елемента матриці на максимальний елемент відповідного рядка. Для кожного рядка визначити середнє значення додатних і від'ємних елементів.

10. Задано матрицю A(n, m). Необхідно упорядкувати рядки матриці по зростанню сум елементів рядків.

11. В числовому масиві A(n, m) знайти суму елементів нижче неголовної діагоналі. Серед них знайти середнє найбільшого і найменшого значення.

12. В заданому цілочисельному масиві Y(n, m) знайти суму і кількість елементів кратним 3 і які стоять на парних позиціях. Серед цих елементів визначити найбільший і найменший елемент.

13. Нехай задано дійсну квадратну матрицю, розміром n на n. Визначити добуток елементів, розмішених вище головної діагоналі. Серед цих елементів визначити мінімальний від'ємний елемент та його координати. Якщо таких елементів немає на дисплей, вивести повідомлення : " Від'ємних елементів немає ".

14. У заданому масиві У(n, m) визначити номер 1-го рядку, який вміщає максимальну кількість додатних елементів. Для знайденого рядка визначити мінімальний і максимальний додатні елементи і їх координати.

15. У масиві X(n, m) визначити суму елементів вище неголовної діагоналі. Серед них визначити мінімальний додатній елемент і його координати. Якщо додатних елементів немає, вивести на екран дисплея повідомлення "Додатних елементів вище неголовної діагоналі немає ".

16. Задано цілочисельну квадратну матрицю A(n, n). Необхідно упорядкувати рядки матриці по зростанню елементів головної діагоналі.

17. Нехай дано квадратну матрицю n*n. Необхідно перетворити матрицю, поелементно відняти останній рядок з усіх рядків, крім останнього. В одержаній матриці упорядкувати рядки по зростанню.

18. Задано цілочисельну квадратну матрицю B(n, n). Необхідно упорядкувати стовпці матриці по зменшенню елементів головної діагоналі.

19. Задано цілочисельну квадратну матрицю C(n, n). Необхідно визначити мінімальні і максимальні елементи, які знаходяться не на діагоналях матриці, а також їх індекси.

20. Задано цілочисельну квадратну матрицю розміром n на n. Знайти номера стовпців, елементи у кожному із яких однакові, а також суму і добуток цих елементів. У випадку відсутності таких елементів вивести на екран дисплея повідомлення " Однакових стовпців немає ".

21. Дано цілочислену квадратну матрицю розміром n на n. Знайти номера рядків, елементи кожного з яких утворюють зростаючу послідовність.

22. Визначити і надрукувати матрицю Z. Яка є добутком матриць X і Y. Результат вивести у вигляді матриці. Вказівка : кожний елемент патриці Z визначається скалярним добутком 1-го рядка матриці X на j- й стовпець матриці Y.

23. Задано матриця B(n, m). Скласти нову матрицю, кожний елемент якої є добутком члена матриці B на мінімальний елемент відповідного стовпця. Одержану матрицю вивести на екран дисплея. Для кожного стовпця визначити середнє значення додатних і від'ємний елементів.

24. Задано матрицю M(n, m). Скласти програму перестановки i та j рядків матриці, де i та задавати у режимі дисплея. Вивести на друк початкову і одержану матриці.

25. Задано матрицю B(n, m). Скласти програму знаходження сум елементів, які знаходяться на лініях, паралельних головній діагоналі патриці і які знаходяться вище неї.

26. Задано матрицю B(n, m). Скласти програму знаходження максимального елемента верхнього трикутника цієї матриці і визначити його координати.

27. Задано матрицю P(n, m). Скласти програму знаходження мінімального елемента матриці. Елементи стовпця і рядка, в яких він знаходиться. замінити нулями. Одержану і початкову матриці вивести на екран дисплея.

28. Задано матрицю M(n, m). Скласти програму знаходження мінімального елемента нижнього трикутника цієї матриці і визначити координати цього елемента.

29. Задано матрицю M(n, m). Скласти програми знаходження сум елементів, які знаходяться на лініях, паралельних головній діагоналі патриці, які знаходяться нижче неї.

30. Розмістити елементи на неголовній діагоналі квадратної матриці A(n,m) в порядку їх зменшення.

РОЗДІЛ 9. РОБОТА З ФУНКЦІЯМИ

9.1. Засоби створення функцій

Функція – це іменована послідовність описів і операторів, що виконує яку-небудь закінчену дію. Функція може приймати параметри і повертати значення.

Будь-яка програма на C++ складається з функцій, головна з яких повинна мати ім'я **main** (з неї починається виконання програми). Функція починає виконуватися у момент виклику. Будь-яка функція має бути оголошена і визначена. Як і для інших величин, оголошень може бути декілька, а визначення тільки одне. Оголошення функції повинне знаходитися в тексті раніше її виклику для того, щоб компілятор міг здійснити перевірку правильності виклику. Оголошення функції (прототип, заголовок) задає її ім'я, тип повертаємого значення та список параметрів, що передаються. Визначення функції містить, окрім оголошення, тіло функції, що є послідовністю операторів і описів у фігурних дужках:

[клас] тип ім'я ([список_параметрів])[throw (виключення)]
{ тіло функції }

Розглянемо складові частини визначення функції.

1. За допомогою необов'язкового модифікатора класу можна явно задати зону видимості функції, використовуючи ключові слова **extern** і **static**:

– **extern** – глобальна видимість у всіх модулях програми (за умовчанням);

– **static** – видимість тільки в межах модуля, в якому визначена функція.

2. Тип повертаємого функцією значення може бути будь-яким, окрім масиву і функції (але може бути вказівкою на масив або функцію). Якщо функція не повинна повертати значення, вказується тип **void**.

3. Список параметрів визначає величини, які потрібно передавати у функцію при її виклику. Елементи списку параметрів розділяються комами. Для кожного параметра, передаваємого у функцію, вказується його тип та ім'я (у оголошенні імена можна не писати).

У визначенні, в оголошенні і при виклику однієї і тієї ж функції типи і порядок параметрів повинні збігатися. На імена параметрів обмежень не накладається, оскільки функцію можна викликати з різними аргументами, а в прототипах імена компілятором ігноруються (вони призначені тільки для покращення читабельності програми).

Функцію можна визначити як вбудовану за допомогою модифікатору **inline**, який рекомендує компілятору замість звернення до функції поміщати її код безпосередньо в кожну точку виклику. Модифікатор **inline** ставиться перед типом функції. Він застосовується для коротких функцій, щоб понизити витрати часу на виклик (збереження та відновлення регістрів, передача управління). Директива **inline** носить рекомендаційний характер і виконується компілятором в міру можливості. Використання **inline**-функцій може збільшити об'єм виконуваної програми. Визначення функції повинне передувати її

викликам, інакше замість **inline**-розширення компілятор згенерує звичайний виклик. Тип повертаємого значення та типи параметрів спільно визначають тип функції.

Для виклику функції в простому випадку потрібно вказати її ім'я, за яким в круглих дужках через кому перераховуються імена аргументів, які передаються. Виклик функції може знаходитися в будь-якому місці програми, де по синтаксису допустимий вираз того типу, який формує функція. Якщо тип повертаємого функцією значення не **void**, вона може входити до складу виразів або, в окремому випадку, розташовуватися в правій частині оператора привласнення.

Розглянемо приклад функції, яка повертає $\log_x y$.

```
#include <iostream>
#include <cmath>
using namespace std;

// Оголошення функції logxy (прототип)
double logxy(double, double);

// Головна функція main
void main()
{
 cout<< logxy(2, 32); //5
}

//Реалізація функції logxy
double logxy(double a, double b)
{
return log(b)/log(a);
}
```

9.2. Видимість змінних

Локальні та глобальні змінні.

Всі величини, описані всередині функції, а також її параметри, є локальними. Зоною їх дії є функція. При виклику функції, як і при вході в будь-який блок, в стеку виділяється пам'ять під локальні змінні. Крім того, в стеку зберігається вміст регістрів процесора на момент, передуючий виклику функції, і адресу повернення з функції для того, щоб при виході з неї можна було продовжити виконання функції, яка визивається.

Локальні змінні визначаються всередині фігурних дужок деякого блоку. Глобальні визначаються поза функцією. Зазвичай глобальні змінні визначаються вище функції main. Всі локальні змінні зникають, коли той блок, у якому вони визначені, завершується. Глобальні змінні доступні від точки їхнього визначення до кінця програми. Так як ви визначаєте глобальну змінну,

то її можна використовувати в будь-якому місці програми нижче її визначення (і не важливо, скільки функцій йде далі).

```
void main()
{
int i, b // локальні змінні
…
}

int g, h // глобальні змінні
void main()
{
…
}
```

Приведемо приклади використання локальних та глобальних змінних.
Приклад 1.

```
#include <iostream>
using namespace std;

float z=10; // глобальна Змінна

void aa(void)
{
  int j=5;
  cout << "\n" << j << " " << z;
  z++;
}

int i=20; // глобальна змінна

void main()
{
  float p=30; z++;
  cout << z << " " << i << " " << p;
  aa();
  cout<< "\n" << z;
}
```

Результат виконання програми:

```
11   20   30
5    11
12
```

Локальна змінна може "жити" (існувати) тільки в межах блоку.

Приклад 2.

```
#include <iostream>
using namespace std;

void main()
{
  int i;
  i=10;
  {
    int i;
    i=20;
    cout << i << " " << i << "\n";
                        // Виведення 20 20
    {
      i=30;
      cout <<i<< "\n"; // Виведення 30
    }
    cout << i<< "\n";  // Виведення 30
  }
  cout << i;           // Виведення 10
}
```

Результат виконання програми:

```
20 20
30
30
10
```

Глобальні змінні видимі у всіх функціях, де не описані локальні змінні з тими ж іменами, тому використовувати їх для передачі даних між функціями дуже легко. Проте це не рекомендується, оскільки ускладнює налагодження програми і перешкоджає розміщенню функцій в бібліотеки загального користування. Потрібно прагнути до того, щоб функції були максимально незалежні, а їх інтерфейс повністю визначався прототипом функції.

Повертаємо значення.

Механізм повертання даних з функції, що викликається, реалізується за допомогою оператора **return [вираз];**

Функція може містити декілька операторів **return** (це визначається потребами алгоритму). Якщо функція описана як **void**, вираз не вказується. Оператор **return** можна не вказувати для функції типу **void**, якщо повернення з неї відбувається перед закриваючою фігурною дужкою, а також для функції **void main**.

Вираз, вказаний після **return**, неявно перетвориться до типу повертаємого функцією значення і передається в точку виклику функції.

Приклади:

```
int f1(){return 1;} //правильно
void f2(){return 1;} // неправильно, f2 не повинна повертати значення
double f3(){return 1;} // правильно, 1 перетвориться до типу double
```

Зверніть увагу, що не можна повертати з функції вказівку на локальну змінну, оскільки пам'ять, виділена локальним змінним при вході у функцію, звільняється після повернення з неї.

Приклад:

```
int* f()
{
 int a = 5;
 return &a; // не можна!
}
```

Автоматичні та статичні змінні.

Ці терміни пояснюють, що відбувається з локальними змінними після того, як управління повертається з функції у функцію, що її викликає.

За умовчанням всі локальні змінні автоматичні, для явного оголошення використовують префікс **auto**. Після блоку (функції) змінні автоматично видаляються.

Локальні статичні змінні не видаляються після завершення функції. Для оголошення статичних змінних використовують ключове слово **static**. Ініціалізація статичних змінних відбувається при першому виконанні функції. Якщо статична змінна не ініціалізується при першому виклику функції, то C++ привласнює їй нульове значення (0).

Приклад:

```
#include <iostream>
using namespace std;

void y(int i)
{
  static int total = 10; //Рядок
  int a;
  a = i*3;
  total += a;
  cout << " " << total;
}

void main()
{
  for(int ctr = 0; ctr < 2; ctr++)
      y(ctr);
}
```

Результат виконання програми:

```
10 13
```

Якщо рядок **static int total = 10**; замінити рядком **static int total**; тоді результат буде: `0 3`

9.3. Параметри функції та передача значень

Механізм передачі параметрів є основним способом обміну інформацією між функціями. Параметри, перераховані в заголовку опису функції, називаються формальними параметрами, або просто параметрами, а записані в операторі виклику функції – фактичними параметрами, або аргументами.

При виклику функції насамперед обчислюються вирази, що стоять на місці аргументів; потім в стеку виділяється пам'ять під формальні параметри функції відповідно до їх типу, і кожному з них привласнюється значення відповідного аргументу. При цьому перевіряється відповідність типів і при необхідності виконуються їх перетворення. При невідповідності типів видається діагностичне повідомлення.

Існує два способи передачі параметрів у функцію: за значенням та за адресою.

При передачі за значенням в стек заносяться копії значень аргументів, і оператори функції працюють з цими копіями. Доступу до початкових значень параметрів у функції немає, а, отже, немає і можливості їх змінити. При передачі за адресою в стек заносяться копії адрес аргументів, а функція здійснює доступ до елементів пам'яті по цих адресах і може змінити початкові значення аргументів:

```
#include <iostream>
using namespace std;

void f(int a, int* b, int& c);

void main()
{
int x = 1, y = 2, z = 3;
cout<<x<<"\t"<<y<<"\t"<<z<<"\n";
f(x, &y, z);
cout<<x<<"\t"<<y<<"\t"<<z<<"\n";
}

void f(int a, int* b, int& c)
{
 a++;
 (*b)++;
 c++;
}
```

Результат виконання програми:

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

Перший параметр (**x**) передається за значенням. Його зміна у функції не впливає на початкове значення. Другий параметр (**y**) передається за адресою за допомогою вказівки, при цьому для передачі у функцію адреси фактичного параметра використовується операція взяття адреси, а для набуття його значення у функції потрібна операція розіменування. Третій параметр (**z**) передається за адресою за допомогою посилання.

При передачі по посиланню у функцію передається адреса вказаного при виклику параметра, а всередині функції всі звернення до параметра неявно розіменовуються. Тому використання посилань замість вказівок покращує читабельність програми, позбавляючи від необхідності застосовувати операції отримання адреси та розіменування. Змінна, яку приймає функція по посиланню є псевдонімом для змінної що передається. У нашому прикладі змінна **c** у функції є псевдонімом до змінної **z**, тобто

```
int& c = z;
```

Використання посилань замість передачі за значенням ефективніше, оскільки не вимагає копіювання параметрів, що має значення при передачі структур даних великого обсягу. Якщо потрібно заборонити зміну параметра всередині функції, використовується модифікатор **const**:

```
int f(const char*);
char* t(char* a, const int* b);
```

Рекомендується вказувати **const** перед всіма параметрами, зміну яких у функції не передбачено. Це полегшує налагодження великих програм, оскільки по заголовку функції можна зробити висновок про те, які величини в ній змінюються, а які ні. Крім того, на місце параметра типу **const&** може передаватися константа, а для змінної при необхідності виконуються перетворення типу.

Таким чином, початкові дані, які не повинні змінюватися у функції, доцільно передавати їй за допомогою константних посилань. За умовчанням параметри будь-якого типу, окрім масиву і функції, передаються у функцію за значенням.

Приведемо приклад обміну двох змінних (стопінг) різними засобами:

```
#include <iostream>
using namespace std;

void swap1(int x, int y);
void swap2(int* x, int* y);
void swap3(int& x, int& y);
```

```
void main()
{
int x = 1, y = 2;
cout<<x<<"\t"<<y<<"\n";

swap1(x, y);
cout<<x<<"\t"<<y<<"\n";

swap2(&x, &y);
cout<<x<<"\t"<<y<<"\n";

swap3(x, y);
cout<<x<<"\t"<<y<<"\n";
}

void swap1(int x, int y)
{
  int z = x;
  x = y;
  y = z; //Обміну в зовнішній програмі не буде.
}

void swap2(int *x, int *y)
{
  int z = *x;
  *x = *y;
  *y = z;
}

void swap3(int &x, int& y)
{
  int z = x;
  x = y;
  y = z;
}
```

Результат виконання програми:

```
1       2
1       2
2       1
1       2
```

При використанні функції swap1 використовується передача аргументів по значенню, отже обміну не буде, стосовно функцій swap2 та swap3 – буде мати місце обмін змінних.

Передача значень за умовчанням.

З метою спрощення виклику функції, в її заголовку можна вказати значення параметрів за умовчанням. Ці параметри повинні бути останніми в списку і можуть не задаватися при виклику функції. Якщо при виклику функції параметр не вказується, мають бути пропущені і всі параметри, що стоять за ним. В якості значення параметрів за умовчанням можуть використовуватися константи, глобальні змінні та вирази:

```
int f1(int a, int b = 0);
void f2(int, int = 100, char* = 0);
// варіанти виклику функції f1
f1(100); f1(a, 1);
// варіанти виклику функції f2
f2(a); f2(a, 10); f2(a, 10, "Vasa");
f2(a, "Vasa") // невірно!
```

Функції із змінним числом параметрів.

Якщо список формальних параметрів функції закінчується багатокрапкою, це означає, що при її виклику на цьому місці можна вказати довільну кількість параметрів. Перевірка відповідності типів для цих параметрів не виконується, char і short передаються як int, a float – як double. Як приклад можна привести функцію printf, прототип якої має вигляд:

```
int printf (const char*, ...);
```

Це означає, що виклик функції повинен містити принаймні один параметр типу char* і може або містити, або не містити інші параметри:

```
printf("Введіть початкові дані"); // один параметр
printf("Сума: %5.2f гривень", sum); // два параметри
printf("%d %d %d", a, b, c); // три параметри
```

Для доступу до необов'язкових параметрів всередині функції використовуються макроси бібліотеки **va_start**, **va_arg**, та **va_end**, що знаходяться в заголовному файлі <**stdarg.h**>.

Оскільки компілятор не має інформації для контролю типів, замість функцій із змінним числом параметрів краще користуватися параметрами за .

9.4. Передача масивів в якості параметрів функцій

При використанні в якості параметру масиву у функцію передається вказівка на його перший елемент, іншими словами, масив завжди передається за адресою. При цьому інформація про кількість елементів масиву втрачається, і слід передавати його розмірність через окремий параметр.

Приведемо приклад обчислення добутку елементів масиву з використанням функції.

```
#include <iostream>
using namespace std;

double proizv(int* mas,int n);
//double proizv(int mas[],int n);

void main()
{
 int mas[4] = {1,2,3,4};
 cout<<proizv(mas,4); //24
}

double proizv(int* mas,int n)
//double proizv(int mas[],int n)
{
 double p = 1;
 for(int i=0;i<n;i++)
     p*=mas[i];

 return p;
}
```

Запис **int* mas** при передачі масиву у функцію еквівалентний запису **int mas[]** та означає вказівку на перший елемент масиву.

При передачі багатовимірних масивів всі розмірності, якщо вони не відомі на етапі компіляції, повинні передаватися в якості параметрів.

Наступний приклад показує роботу з двовимірним масивом як з вказівкою **int (*mas)[3]**.

```
#include <iostream>
using namespace std;

void vivod(int (*mas)[3], int m, int n)
//void vivod(int mas[][3], int m, int n)
{
 int i,j;
 for(i=0;i<m;i++)
  {
     for(j=0;j<n;j++)
       cout<<mas[i][j]<<" ";
          cout<<"\n";
   }
}
```

```
void main()
{

 int mas[3][3]=
                                {
                        {11,12,13},
                        {21,22,23},
                        {31,32,33}
                                 };

int(*m)[3]= mas;

vivod(++m, 2, 3);

}
```

По суті, двовимірний масив є вказівкою **int(*m)[3]**. Це означає, що при арифметичних операціях з вказівкою, вона збільшується або зменшується на кількість байтів трьох елементів типу даних **int** до першого розмикання. Після першого розмикання адреса збільшується на кількість байтів одного елементу **int**, як у разі звичайної вказівки. У даному прикладі вказівці **int(*m)[3]** привласнюється значення вказівки двовимірного масиву **mas**. Потім, у функцію **vivod** передається вказівка **m** збільшена на одиницю. Функція **vivod** набуває значення вказівки та виводить другий і третій рядки масиву:

```
21 22 23
31 32 33
```

При передачі у функцію записи **int (*mas)[3]** та **int mas[][3]** еквівалентні. Другий параметр у дужках вказувати обов'язково, саме він говорить на скільки елементів масиву треба зміщуватись при виконанні арифметичних операцій.

Можна передати двовимірний масив у функцію як подвійну вказівку ("**").

Приклад.

```
#include <iostream>
using namespace std;

void vivod(int **mas, int m, int n)
{
 int i,j;
 for(i=0;i<m;i++)
  {
     for(j=0;j<n;j++)
       cout<<mas[i][j]<<" ";
          cout<<"\n";
   }
}
```

```
void main()
{
 //m, n - відповідно кількість рядків
 // і стовпців масиву

 int i, j, m, n;
 cin>>m>>n;

//Ініціалізація динамічного масиву
 int **mas =   new int*[m];
 for (i=0;i<m;i++)
        mas[i]= new int[n];

 //Введення даних
 for(i=0;i<m;i++)
     for(j=0;j<n;j++)
    cin>>mas[i][j];

 //Виведення даних
 vivod(mas, m, n);

 //Видалення динамічного масиву
 for (i=0;i<m;i++)
    delete[] mas[i];

 delete [] mas;
}
```

Спочатку оголошується змінна типу «вказівка на вказівку на **int**» і виділяється пам'ять під масив вказівок на рядки масиву (кількість рядків – **m**). Потім організовується цикл для виділення пам'яті під кожен рядок масиву. Кожному елементу масиву вказівок на рядки привласнюється адреса початку ділянки пам'яті, виділеної під рядок двовимірного масиву. Кожен рядок складається з **n** елементів типу **int**. Звільнення пам'яті двовимірного масиву також відбувається в два етапи. Спочатку в циклі видаляються **m** динамічних рядків, кожен з яких є вказівкою на одновимірний масив. Потім видаляється сам масив вказівок на рядки.

9.5. Функції та вказівки

Вказівка на функцію містить адресу в сегменті коду, по якій розташовується виконуваний код функції, тобто адресу, по якій передається управління при виклику функції. Вказівки на функції використовуються для непрямого виклику функції (не через її ім'я, а через звернення до змінної, що зберігає її адресу), а також для передачі імені функції в іншу функцію як параметр. Вказівка функції має тип «вказівка функції, що повертає значення

заданого типу і що має аргументи заданого типу»:

тип (*ім'я) (список_типів_аргументів);

Наприклад, оголошення:

```
int (*fun) (double, double);
```

задає вказівку з ім'ям **fun** на функцію, що повертає значення типу **int** і що має два аргументи типу **double**.

Приведемо простий приклад, в якому оголошується вказівка на функцію і привласнюється адресі функції (ім'я функції є її адресою)

```
#include <iostream>
using namespace std;

void vivod(int x)
{
 cout<<"x = "<<x;
}

void main()
{
 vivod(2);
 cout<<"\n";

 cout<<"vivod = "<<vivod;

 void (*ff)(int);

 ff = vivod;

 cout<<"\n";
 ff(25);
}
```

У даному прикладі функція **vivod** просто виводить змінну типу **int** на екран дисплея. Потім виведемо на екран адресу функції **vivod**:

```
cout<<vivod;
```

На наступному етапі оголосимо вказівку на функцію з вхідним параметром **int** і поверненням **void**:

```
void (*ff)(int);
```

Після рядка привласнення ff = vivod, адресою на функцію **ff** можна користуватися також як і функцією **vivod**. Результатом виконання програми будуть наступні рядки:

x = 2
vivod = 0x0ce5000b
x = 25

Передача імен функцій в якості параметрів.

Функцію можна викликати через вказівку на неї. Для цього оголошується вказівка відповідного типу і їй привласнюється адреса функції;

```
void f(in t a ){/*...* / } // визначення функції
void (*pf)(int); // вказівка на функцію
pf = f; // вказівці привласнюється адреса функції
pf(10); // функція f викликається через вказівку pf
            // (можна написати (*pf)(10) )
```

Для того, щоб програму було легко прочитати, при описі вказівок на функції використовують перевизначення типів (**typedef**). Можна оголошувати масиви вказівок на функції (це може бути корисно, наприклад, при реалізації меню);

```
// Опис типу PF як вказівки
// на функцію з одним параметром типу int;
typedef void (*PF)(int);
// Опис і ініціалізація масиву вказівок:
PF menu[] = {&new. &open. &save};
menu[l](10); // Виклик функції open
```

Тут new, open і save – імена функцій, які мають бути оголошені раніше.

Вказівки на функції передаються в підпрограму таким же чином, як і параметри інших типів:

```
#include <iostream>
using namespace std;

typedef void (*PF)(int);

// функція f1 отримує як параметр вказівку типу PF
void f1 (PF pf)
{
 //виклик функції, переданої через вказівку
 pf(5);
}
void f(int i)
{
 cout<<i;
}

void main()
{
 f1(f);
}
```

Тип вказівки і тип функції, яка викликається за допомогою цієї вказівки, повинні збігатися в точності.

Потреба в передачі вказівки функції як параметр іншої функції виникає при рішенні задачі про побудову таблиці значень функції. У наступному прикладі представлена програма, що будує таблицю значень довільної функції.

```
#include <iostream>
using namespace std;

typedef double (*function_type)(double);
void form_table(function_type f,
                double low, double up, double inc)
{
 double x;
 x = low;
 printf("x                    y\n");

 while(x-0.1*inc<=up)
 {
  printf("\n%-20.5f %-20.5f",x,f(x));
  x+=inc;
 }
}

double my_function(double x)
{
 return x*x;
}

void main()
{
 form_table(my_function,0.0,2.0,0.1);
}
```

Перший параметр **f** функції побудови таблиці **form_table** має тип **function_type**. Так задається початкова функція, значення якої мають бути поміщені в таблицю. Другий і третій параметри вказують область визначення функції. Четвертий параметр задає інтервал (крок) між сусідніми значеннями аргументу функції. У даному прикладі будується таблиця для квадратичної функції **my_function**. В результаті виконання програми на екран дисплея буде виведена наступна таблиця:

```
x                    y
0.00000              0.00000
0.10000              0.01000
0.20000              0.04000
0.30000              0.09000
```

```
0.40000                    0.16000
0.50000                    0.25000
0.60000                    0.36000
0.70000                    0.49000
0.80000                    0.64000
0.90000                    0.81000
1.00000                    1.00000
1.10000                    1.21000
1.20000                    1.44000
1.30000                    1.69000
1.40000                    1.96000
1.50000                    2.25000
1.60000                    2.56000
1.70000                    2.89000
1.80000                    3.24000
1.90000                    3.61000
2.00000                    4.00000
```

9.6. Перевантаження та шаблони функцій

Перевантаження функцій.

Для реалізації одного й того ж алгоритму можна використовувати функції, які мають одне й теж ім’я. Використання декількох функцій з одним й тим же ім’ям, але з різними типами параметрів, називають перевантаженням функцій.

По типу вхідних та вихідних параметрів компілятор визначає, яку функцію треба визвати.

Наприклад, треба знайти мінімальне значення з двох чисел.

```
#include <iostream>
using namespace std;

int min(int x, int y)
{
 cout<<"min1 = ";
 return (x<y)?x:y;
}

double min(double x, double y)
{
 cout<<"min2 = ";
 return (x<y)?x:y;
}

void main()
{
 cout<<min(3, 4)<<"\n";
```

```
 cout<<min(3.0, 4.0)<<"\n";
}
```

Результат виконання програми:

```
min1 = 3
min2 = 3
```

Тобто у першому випадку спрацювала функція, яка приймає та повертає значення типу **int**, у другому випадку – **double**.

Перевантажені функції можуть мати параметри за умовчанням, при цьому значення одного й того ж параметру у різних функціях повинні мати однакове значення. Функції не можуть бути перевантажені, якщо опис їх параметрів відрізняється тільки модифікатором **const** або використанням посилання (наприклад **int** та **const int** або **int** та **int&**).

Шаблони функцій.

За допомогою шаблонів можна створювати родові функції (**generic functions**). У родовій функції тип даних, з яким функція працює, задається у якості параметру. Це дозволяє одну й ту ж функцію використовувати з декількома різноманітними типами даних та без необхідності програмувати нову версію функції або класу для кожного конкретного типу даних. Таким чином шаблони дають можливість створювати багатократно використовуємі програми.

Родова функція визначає базовий набір операцій, які будуть застосовуватись до різних типів даних. Родова функція працює з тим же типом даних, який вона отримує у якості параметру. Як відомо, багато алгоритмів логічно однакові, незалежно від того, для обробки яких типів даних вони призначені. Наприклад, алгоритм швидкого сортування однаковий як для масивів цілих, так і для масивів дійсних чисел. Це той випадок, коли данні, що сортуються відрізняються тільки по типам. За допомогою створення родової функції можна незалежно від типу даних визначити сутність алгоритму. Після того як це зроблено, компілятор автоматично генерує правильний код для фактично використовуваного при виконанні функції типу даних. При створенні родової функції – створюється функція, яка може автоматично перевантажуватися сама.

Родова функція створюється за допомогою ключового слова **template**. Нижче представлено типову форму визначення функції-шаблону:

template <class Фтип> повертаєме_значення ім’я_функції (список параметрів)
{
//тіло функції
}

Тут замість ***Фтип*** вказується тип використовуваних функцією даних. Цей тип можна вказувати усередині визначення функції. Однак, він є фіктивним,

його компілятор автоматично замінить реальним типом даних при створенні конкретної версії функції.

Наведемо приклад програми в якій створюється родова функція, яка міняє місцями значення двох змінних, що передаються їй в якості параметрів. Оскільки процес обміну двох значень не залежить від типу змінних – його реалізація ефективна за допомогою родової функції.

```
#include <iostream>
using namespace std;

//Функція-шаблон
template <class A> void swap1(A&x, A&y)
{
 A temp;
 temp = x;
 x = y;
 y = temp;
}

void main()
{
 int a, b;
 a = 5; b = 10;
 double x, y;
 x = 5.8; y = 10.27;

 cout<<"a = "<<a<<"  b = "<<b<<"\n";
 cout<<"x = "<<x<<"  y = "<<y<<"\n";

 swap1(a, b); //обмін цілих чисел
 swap1(x, y); //обмін дійсних чисел

 cout<<"a = "<<a<<"  b = "<<b<<"\n";
 cout<<"x = "<<x<<"  y = "<<y<<"\n";
}
```

Результат виконання програми наступний:

```
a = 5        b = 10
x = 5.8      y = 10.27
a = 10   b = 5
x = 10.27    y = 5.8
```

Ключове слово **template** використовується для визначення родової функції. Рядок:

```
template <class A> void swap1(A&x, A&y)
```

повідомляє компілятору дві речі: по-перше, створюється шаблон, та по-друге, починається визначення родової функції. Тут **A** – родовий тип даних. Після рядка з ключовим словом **template** функція **swap()** оголошується з типом даних **A** значень, що обмінюються. У функції **main()** функція **swap1()** викликається з двома різними типами даних: цілими та дійсними. Оскільки функція **swap1()** – родова функція, компілятор автоматично створює дві її версії: одну для обміну цілих значень, іншу для обміну дійсних значень.

За допомогою інструкції **template** можна визначити більше одного родового типу даних. Наприклад, у даній програмі створюється родова функція, в якій є два родові типи даних:

```
#include <iostream>
using namespace std;

//Функція-шаблон
template <class type1, class type2>
void myfunc(type1 x, type2 y)
{
 cout<<"x = "<<x<<"  y = "<<y<<"\n";
}

void main()
{
 myfunc(10,"Hello");
 myfunc(5.15,13L);
}
```

Результат виконання програми наступний:

```
x = 10   y = Hello
x = 5.15     y = 13
```

У даному прикладі при генерації конкретних екземплярів функції **myfunc()**, типи даних **type1** та **type2** замінюються компілятором на типи даних **int** та **char*** або **double** та **long** відповідно.

Контрольні питання

1. Що таке функція?
2. Розкрийте сутність глобальних та локальних змінних.
3. Чим відрізняються автоматичні та статичні змінні?
4. Які ви знаєте способи передачі змінних до функції?
5. В чому відмінність передачі змінних по значенню та за посиланням?
6. Для чого потрібен прототип функції?
7. Як передавати значення змінних функції за умовчанням?
8. Розкрийте сутність статичних змінних.

9. Наведіть приклад передачі змінних по значенню.
10. Наведіть приклад передачі змінних за посиланням.
11. Яким чином передати одновимірний масив до функції?
12. Яким чином передати двовимірний масив до функції?
13. Що таке вказівка на функцію?
14. Як передати ім'я (адресу) функції у якості параметру до іншої функції?
15. Застосування перевизначення типів typedef для опису вказівок на функції.
16. Наведіть приклади передачі вказівок функції.
17. Що таке перевантаження функцій?
18. Наведіть приклади перевантаження функцій.
19. Що таке шаблони функцій?
20. Наведіть приклади використання шаблонів функцій.

Завдання

Скласти програму з використанням функцій. Якщо функції мають обмежену область визначення, дослідити їх та зробити контроль вхідних даних.

Приклад виконання завдання.

Скласти програму з використанням функції $y = 2(\ln(x^2 + 2) + 1) \cdot \sin x$.

Код програми має наступний вигляд:

```
#include <iostream>
#include <cmath>
using namespace std;

double func(double x)
{
 return 2*(log(x*x+2)+1)*sin(x);
}

void main()
{
 double x,y;
 cout<<"Введите значение x ";
 cin>>x;
 y = func(x);
 cout<<"\ny = "<<y;
}
```

Варіанти індивідуальних завдань

1. $Y = (\log_2 x + \log_e z)/[2\log_{x+2}(x+z)]$;
2. $Y = k \cdot x \cdot e^{ax} / (1+bx)^2$;

3. $Y = \frac{x^2}{2} \cdot tg\frac{x^2}{2} \Big/ \left(1 + \frac{x^2}{2}\right);$

4. $Y = \sqrt{a^2 + b^2 + c^2} \Big/ \sqrt{n^2 + m^2 + k^2}\ ;$

5. $Y = \ln\left(1 + a\sqrt{1 + a^2}\right) \Big/ \left(b\sqrt{1 + b^2}\right);$

6. $Y = e^x \left(1 + \cos 2a\right) / \left(1 + \cos 2b\right);$

7. $Y = \operatorname{arctg}\left(\frac{x - a}{\ln a}\right) \Big/ \left[1 + \sin\left(\frac{x - b}{\ln b}\right)\right];$

8. $Y = \frac{\ln^2\left(a - x^2\right)}{b^2 - \ln^2\left(a - x^2\right) - \ln^6\left(a - x^2\right)};$

9. $Y = \frac{x \log_3 x - z \log_5 z}{a \log_b a - b \log_a b};$

10. $Y = \sqrt[3]{\left(a + \ln a\right)^2} \Big/ \sqrt{b + \ln b}\ ;$

11. $Y = \frac{\frac{x - \ln a}{\ln^2 x} \sin\left(\frac{z - \ln a}{\ln^2 z}\right)}{\sqrt[3]{\frac{b - \ln a}{\ln^2 b}}};$

12. $Y = e^{-\frac{x^2 - a^2}{\sin x}} \cdot \sqrt[5]{\frac{z^2 - a^2}{\sin z}}\ ;$

13. $Y = \frac{\ln^2 ax - 1}{2 \sin x} \Big/ \frac{\ln^2 bx - 1}{3 \cos x};$

14. $Y = \left(a^{2,1} - \sin ax\right) / \left(b^{2,1} - \sin bx\right);$

15. $Y = \sqrt{\log_7\left(x^2 + 5\right)} \cdot \sqrt[3]{\log_9\left(x^2 + 7\right)};$

16. $Y = n \cdot k \cdot x^{n-1} \cdot e^{-k \cdot x^n}\ ;$

17. $Y = \frac{\ln\left(x^2 + a\sqrt{1 + x^2 + x^4}\right)}{b\sqrt{1 + z^2 + z^4}};$

18. $Y = \frac{ax + z \cdot \left(z + \sqrt{\left|1 + x + 2x^2 + 3x^3\right|}\right)}{z - x + \sqrt{\left|1 + z + 2z^2 + 3z^3\right|}};$

19. $Y = \frac{a}{b\sqrt{a^2 + b^2}} \cdot \operatorname{arctg}\frac{ax}{\sqrt{x^2 + a^2}};$

20. $Y = \left(1 + x^2 + z^2\right) / \left(3 + \sqrt{x^2 + z^2}\right);$

21. $Y = \frac{1}{3}\pi \cdot \left(a^2 + b^2 + ab\right) \cdot x\ ;$

22. $Y = \frac{ax - bz\left(x^2 + \sqrt[3]{1 + x^2 + 2x^4 + 3x^6}\right)}{\sin c \sqrt[3]{1 + \sin^2 c + 2\sin^4 c + 3\sin^6 c}};$

23. $Y = e^{\cos kx^2} \cdot \dfrac{ax}{b \cdot \cos az}$;

24. $Y = \operatorname{tg} \dfrac{ax^2 - b}{2a} - \operatorname{ctg} \dfrac{bz^2 - a}{2b}$;

25. $Y = \dfrac{\sin\left(\sqrt[5]{x^3} + 1\right)}{\sqrt[5]{z^3} + 1} \cdot \cos^2\left(\dfrac{\sqrt[5]{x^3} + 1}{\sqrt[5]{z^3} + 1}\right)$;

26. $y = \sum\limits_{k=1}^{n} \dfrac{k!}{n!}$;

27. $y = \sum\limits_{k=1}^{n} \dfrac{(m+n)!}{k!}$;

28. $y = \sum\limits_{k=1}^{n} \dfrac{k!}{(n+m+k)!}$;

29. $y = \sum\limits_{k=1}^{n} \dfrac{(m-n)!}{(m+k)!}$;

30. $y = \sum\limits_{k=4}^{m-n} \dfrac{m! - n!}{k!}$.

РОЗДІЛ 10. РЕКУРСИВНЕ ПРОГРАМУВАННЯ

10.1. Основні поняття рекурсії. Визначення факторіалу числа

Рекурсивною називається функція, яка викликає саму себе. Така рекурсія називається прямою. Існує ще непряма рекурсія, коли дві або більше функцій викликають одна одну. Якщо функція викликає себе, в стеку створюється копія значень її параметрів, як і при виклику звичайної функції, після чого управління передається першому виконуваному операторові функції. При повторному виклику цей процес повторюється. Ясно, що для завершення обчислень кожна рекурсивна функція повинна містити хоч би одну нерекурсивну гілку алгоритму, що закінчується оператором повернення. При завершенні функції відповідна частина стека звільняється, і управління передається викликаємій функції, виконання якої продовжується з точки, наступної за рекурсивним викликом.

Класичним прикладом рекурсивної функції є обчислення факторіалу (це не означає, що факторіал слід обчислювати саме так). Для того, щоб отримати значення факторіалу числа n, потрібно помножити на n факторіал числа (n-1). Відомо також, що 0!=1 і 1!=1.

```
long fact (long n)
{
if (n==0 || n==1) return 1;
return (n * fact(n - 1));
}
```

Те ж саме можна записати коротше:

```
long fact (long n)
{
return (n>1) ? n * fact(n - 1): 1;
}
```

Рекурсивні функції найчастіше застосовують для компактної реалізації рекурсивних алгоритмів, а також для роботи із структурами даних, описаними рекурсивно, наприклад, з двійковими деревами. Рекурсивну функцію можна реалізувати без застосування рекурсії, для цього потрібно забезпечити зберігання всіх необхідних даних самостійно. Перевагою рекурсії є компактний запис, а недоліками – витрати часу і пам'яті на повторні виклики функції і передачу їй копій параметрів, і, головне, небезпека переповнення стека.

10.2. Приклади рекурсій

Розглянемо рекурсивну функцію, що обчислює n-й член послідовності чисел Фібоначчі. Ці числа визначаються рекурентним співвідношенням

$f_n = f_{n-1} + f_{n-2}$ для $n > 0, f_1 = 1, f_0 = 0$.

Приведемо програмну реалізацію даної функції.

```
#include <iostream>
using namespace std;

long Fib(long n)
{
 if (n<2)return n;
 else return Fib(n-1)+Fib(n-2);
}

void main()
{
 int i;
 for (i=0;i<=5;i++)
     cout<<Fib(i)<<"\t";
}
```

У результаті буде виведений наступний ряд чисел Фібоначчі:

0 1 1 2 3 5

Кожен виклик функції Fib при n>1 породжує два вкладених виклики. У результаті відбуваються повторні обчислення тих самих величин і загальна кількість вкладених викликів зростає експоненціально. Цей процес ілюструє рис. 10.1, на якому числа відповідають значенням параметра n.

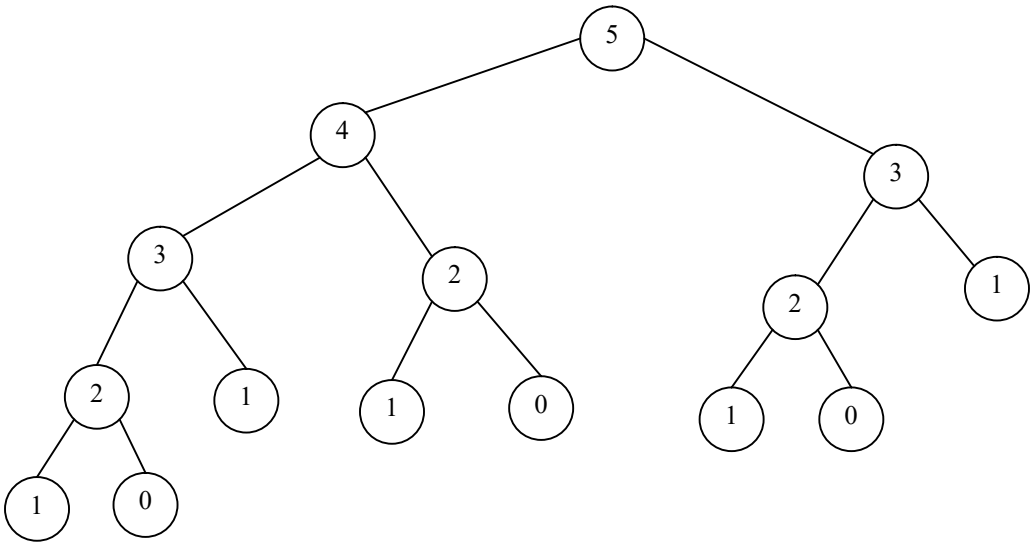


Рис.10.1. Вкладені рекурсивні виклики функції, що повертає число Фібоначчі

Наприклад, виконання функції Fib(5) призведе до того, що виклик Fib(3) буде здійснено двічі, виклики Fib(2) та Fib(0) – тричі, а виклик Fib(1) – п’ять разів. Загальна кількість вкладених викликів сягне п’ятнадцяти. Тому для практичного використання така програма непридатна.

«Індійський алгоритм» піднесення числа x до натурального степеня n реалізує таке рекурсивне означення степеня числа:

$$x^n = \begin{cases} 1, & n = 0; \\ x, & n = 1; \\ x^{n\%2} \cdot (x^{n/2})^2, & n > 1. \end{cases}$$

Тривіальний спосіб піднесення числа x до степеня n потребує, щоб виконувались $n - 1$ операцій множення: x множиться на x, добуток знову множиться на x, і так $n - 1$ разів. Застосування індійського алгоритму дає можливість значно скоротити кількість операцій. Наприклад, обчислення x^{10} за індійським алгоритмом потребує лише чотирьох операцій множення замість дев'яти. Справді, $x^{10} = (x^5)^2 = (x \cdot (x^2)^2)^2$. Тобто, при обчисленні x^{10} операції виконуватимуться в такому порядку:

1) обчислюємо x^2;
2) x^2 підносимо до квадрата і отримуємо x^4;
3) x^4 множимо на x і отримуємо x^5;
4) x^5 підносимо до квадрата і отримуємо x^{10}.

Наведене вище рекурсивне означення степеня числа може бути реалізоване рекурсивною функцією. При рекурсивному зануренні така функція обчислюватиме значення $n/2$, а при виході з рекурсії обчислюватиметься множник $x^{n\%2}$ і виконуватиметься операція множення. Оскільки $n\%2$ дорівнює нулю, якщо n – парне, і дорівнює одиниці, якщо n – непарне, то величина $x^{n\%2}$ дорівнюватиме 1 за парних значень n і дорівнюватиме x за непарних значень n. Отже, наведемо деталізований індійський алгоритм піднесення до степеня та його програмну реалізацію.

Алгоритм обчислення x^n

1. Якщо показник степеня дорівнює нулю, то значення x^n покласти рівним одиниці та вийти з рекурсії, інакше виконати дії, зазначені в пунктах 2–5.

2. Якщо показник степеня дорівнює одиниці, то значення x^n покласти рівним x та вийти з рекурсії, інакше виконати дії, зазначені в пунктах 3–5.

3. Обчислити значення $x^{n/2}$ та піднести його до квадрата.

4. Якщо показник степеня непарний, то обчислене в пункті 3 значення помножити на число x, а отриманий результат вважати значенням x^n.

5. Інакше, коли показник степеня парний, то обчислене в пункті 3 значення вважати значенням x^n.

Наведемо програмну реалізацію індійського алгоритму.

```
#include <iostream>
using namespace std;

//Функція піднесення x до степеня n
long powxn(long x, long n)
{
 if(n==0) return 1;
 if(n==1) return x;

 long t;
```

```
 t = powxn(x,n/2)*powxn(x,n/2);
 if (n&1)return t*x;
 else return t;
}

void main()
{
 cout<<"2^8 = "<<powxn(2,8);
}
```

У результаті виконання програми отримаємо:

```
2^8 = 256
```

Тепер опишемо залежність глибини рекурсії від значення степеня *n*. У кожному наступному вкладеному виклику значення аргументу *n* менше від попереднього значення принаймні вдвічі. Оскільки за умови *n* = 1 розпочинається рекурсивне повернення, то таких зменшень значення аргументу *n* не може бути більше $\log_2 n$. Отже, глибина рекурсії не перевищує $\log_2 n$. Таку глибину можна вважати ознакою високої ефективності алгоритму. Під час виконання кожного рекурсивного виклику відбувається не більше ніж одна операція ділення, піднесення до квадрата та множення, тому загальна кількість арифметичних операцій не перевищує $3\log_2 n$. Якщо величина *n* достатньо велика, це суттєво менше, ніж *n*–1 множень. Наприклад, за умови *n* = 1000 кількість арифметичних операцій приблизно дорівнює 30.

Контрольні питання

1. Що таке рекурсія?
2. Переваги та недоліки рекурсивного програмування.
3. Наведіть власний приклад рекурсії та проаналізуйте глибину рекурсії.
4. Наведіть приклад рекурсії розрахунку факторіалу числа.
5. Наведіть приклад рекурсії розрахунку n-го члену послідовності Фібоначчі.
6. Наведіть алгоритм піднесення числа x до натурального степеня n.

РОЗДІЛ 11. РОБОТА З ФАЙЛАМИ

11.1. Робота з текстовими та бінарними файлами

Файли бувають двох типів: текстовими та бінарними (binary).

Доступ до інформації у файлах довільний, тобто звертання до файлів може здійснюватись в будь-якому порядку.

Відкриття і закриття файлів

Для відкриття використовується функція-член **open()** з одним з об'єктів файлових потоків. Для закриття файлів використовується функція-член **close():**

Формат:

fs_obj.open(file_name, access);
fs_obj. close();

fs_obj – об'єкт, що являє собою файл.

file_name – рядок (або вказівка на рядок); може бути зазначений весь маршрут.

access – являє собою будь-яке значення з таблиці можливих режимів доступу (табл. 11.1) або комбінацію з використанням оператора **АБО** (|)

Таблиця 11.1

Можливі режими доступу

Режим	Опис	Код
ios::in	Відкрити файл для читання	1
ios::out	Відкрити файл для запису	2
ios::ate	Відкрити файл і встановити вказівку на кінець	4
ios::app	Відкрити файл для додавання	8
ios::trunc	Видалити уміст файла	16
ios::nocreate	Якщо немає файла, повертає відмову	32
ios::noreplace	Якщо файл є, його відкриття тільки для додавання	64
ios::binary	Відкрити файл у двоїчному режимі	128

Якщо ви відкриваєте файл для запису з опцією **ios::out**, C++ просто створює його. Якщо файл із цим ім'ям вже існує, C++ без усякого попередження перекриває старий файл, знищуючи його вміст.

Якщо ви хочете, щоб спроба відкрити файл tt.txt завершилася відмовою, у випадку, якщо файл не існує, використовуйте:

```
cur_fs.open("tt.txt",ios::out | ios::nocreate)
```

Режим доступу до файлів по умовчанню – текстовий ASCII-файл.

Бінарний режим доступу до файлів більш компактний, дані записуються в двійкових машинних кодах.

Скажімо

```
int i = 31764;
```

для ASCII - коду – це 5 символів,
для двійкового машинного коду – це 2 байта.

Якщо під час відкриття файла відбувається помилка, об’єкт файлового потоку буде дорівнювати нулю.

Наприклад, якщо ви відкриваєте файл з опцією ios::nocreate і заданий файл не існує, C++ не відкриває цей файл. Виникнення помилки можна простежити у такий спосіб:

```
if(! cur_fs) cout << "Файл не існує";
```

Рекомендації: Краще відкривати файли перед їхнім використанням і закривати відразу ж після використання.

Запис у файл.

Найбільш розповсюдженні функції файлового введення-виведення:

- **get()**;
- **put()**.

Приклади:

Приклад 1.

```
#include <fstream>
using namespace std;

ofstream fp;

void main()
{
  fp.open("nam.dat", ios::out);

  fp<< "Петров Сергій\n";
  fp<< "Сидоров Іван\n";

  fp.close();
}
```

При перегляді файлу "nam.dat" отримаємо наступну інформацію

```
Петров Сергій
Сидоров Іван
```

Приклад 2. Запис у файл цифр від 1 до 100

```
#include <iostream>
#include <fstream>
using namespace std;
```

```
ofstream fp;

void main()
{
  int i;
// Створення нового файла
  fp.open("nam.dat", ios::out);

  if(!fp)
    cout << "Помилка відкриття файла\n";
  else
  {
    for(i = 1; i < 101; i++)
      fp << i << " ";
  }
  fp.close();
}
```

Виведення на принтер.

Принтер, також як і екран, можна розглядати як аналогію з файлом. Приклад:

```
#include <fstream>
using namespace std;

ofstream prn;

void main()
{
  prn.open("LPT1",ios::out);
  prn << "Міша\n";
  prn << "Петро\n";
  prn.close();
}
```

Результат виконання програми:

```
Міша
Петро
```

Додавання у файл і читання з файла.

Приклад. Додати 3 прізвища у файл, у котрому вже є 2 прізвища:

Петров
Сидоров

```
#include <fstream>
using namespace std;

ofstream fp;

void main()
{
  fp.open("name.dat", ios::app);
  fp << "Козлов\n";
  fp << "Ромін\nКравенко\n";
  fp.close();
}
```

Якщо файл **name.dat** не існує, то створюється новий із трьома прізвищами.

Приклад. Програма запитує в користувача ім’я файла і виводить його вміст на екран.

```
#include <iostream>
#include <fstream>
using namespace std;

fstream fp;

void main()
{
  char filename[12];
  char in_char;
  cout << "Введіть ім’я файла ";
  cin >> filename;
  fp.open(filename, ios::in);
  if(!fp)
  {
    cout << "\nПомилка, файл не відкривається\n";
    exit(0); //Вихід із програми
  }
  while(fp.get(in_char))
    cout << in_char;
  fp.close();
}
```

Приклад. Копіювання даних з одного файла в інший:

```
#include <iostream>
#include <fstream>
using namespace std;
```

```
ifstream in_fp;
ofstream out_fp;

void main()
{
  char in_char;
  char in_file[12];
  char out_file[12];
  cout << "Введіть ім'я початкового файлу ";
  cin >> in_file;

  cout << "Введіть ім'я вихідного файлу ";
  cin >> out_file;

  in_fp.open(in_file, ios::in);
  if(!in_fp)
  {
    cout << "\n\nПочаткового файлу не існує\n";
    exit(0);
  }
  out_fp.open(out_file, ios::out);
  if(!out_fp)
  {
    cout << "\n\nПомилка відкриття вихідного файлу\n";
    exit(0);
  }
  cout << "\nКопіювання\n";
  while(in_fp.get(in_char))
    out_fp.put(in_char);
  out_fp.close();
  in_fp.close();
}
```

11.2. Довільний доступ у файлах

Більшість файлів складаються з фіксованого числа записів. Це дозволяє звернутися до будь-якого запису.

Якщо файл створюється та в нього записуються дані, а потім його необхідно скорегувати, тобто виконати операції введення-виведення, необхідно його відкривати таким чином:

```
fp.open("nam.txt", ios::in | ios::out);
if(!fp)
  . . .
```

У випадку якщо файлу не існує він не буде створюватись та змінна **fp** буде дорівнювати нулю.

Якщо такі ж дії виконуються з бінарним файлом, тоді

```
fp.open("nam.txt",ios::in|ios::out|ios::binary);
  . . .
```

Використання функції seekg().

Ця функція забезпечує зчитування даних, починаючи з будь-якого місця файлу.

Формат:

fs_obj. seekg(long_num, origin);

long_num – число байтів у файлі, які треба пропустити, тобто зчитування або запис даних виконується з заданої позиції.

Оскільки файли даних мають великий розмір, необхідно оголосити **long_num** з типом **long int**.

origin – є значенням, що повідомляє C++, звідки необхідно почати пропускати байти, зазначені в **long_num**. Ця змінна може приймати одне з трьох значень:

Режим	**Опис**
ios::beg	Початок файла
ios::cur	Поточна позиція курсору
ios::end	Кінець файла

Індикатори режиму **ios::beg, ios::cur, ios::end** визначені в файлі <**iostream**>

Для того щоб визначити поточну позицію курсору в файлі використовується функція **tellg()**, яка повертає значення типу **long**.

Приклад 1.

Файл читається двічі, перший раз для виведення на екран, другий - на принтер.

```
#include <iostream>
#include <fstream>
using namespace std;

ifstream in_file;
ofstream scr;
ofstream prn;

void main()
{
  char in_char;
  in_file.open("name.txt", ios::in);
  if(!in_file)
  {
    cout << "\nПомилка відкриття файлу\n";
    exit(0);
```

```
  }
  scr.open("CON", ios::out);
  while(in_file.get(in_char))
    scr << in_char;
  scr.close();

  in_file.clear();

  in_file.seekg(0L, ios::beg);
  prn.open("LPT1", ios::out);
  while(in_file.get(in_char))
    prn << in_char;
  prn.close();
}
```

Після виведення вмісту файлу на екран дисплею ("CON") вказівка переводиться на кінець файлу та виникає біт помилки **ios::eofbit**, що дорівнює 1. Для того щоб продовжити роботу з файлом потоку, необхідно скинути цей біт. Робиться це за допомогою функції **clear()**, яка скидує всі біти помилок файлового потоку.

Приклад 2.

Замінити літери **i** та **q**, що займають у тексті файлу 9 і 17 позиції на букву **X**.

```
#include <fstream>
using namespace std;

fstream fp;

void main()
{
  char ch;
  fp.open("alph.txt", ios::out);
  for(ch = 'a'; ch <= 'z'; ch++)
  fp << ch;
  fp.seekg(8L, ios::beg);
  fp << 'X';
  fp.seekg(16L, ios::beg);
  fp << 'X';
  fp.close();
}
```

Вміст файлу **alph.txt** буде виглядати наступним чином:

```
abcdefghXjklmnopXrstuvwxyz
```

Приклад 3.

Прочитати файл **alph.txt** із попереднього прикладу і вивести на екран в зворотньому порядку.

```
#include <iostream>
#include <fstream>
using namespace std;

ifstream fp;

void main()
{
  int ctr;
  char inchar;
  fp.open("alph.txt", ios::in);
  fp.seekg(-1L, ios::end);
  for(ctr = 0; ctr <= 25; ctr++)
  {
    fp >> inchar;
    fp.seekg(-2L, ios::cur);
    cout << inchar;
  }
  fp.close();
}
```

Результат виконання програми:

```
zyxwvutsrXponmlkjXhgfedcba
```

Використання додаткових функцій роботи з файлами.

read(char* array, int count) – зчитує кількість байтів, визначуваних у **count** в масив **array**.

write(char* array, int count) – записує масив із кількістю даних **count** у файл.

int remove(char* filename) – видаляє файл з ім’ям **filename**.

Приклади програм роботи з файлами

Приклад 1. Записати масив у файл **ot.txt**, прочитати його і вивести на екран.

```
#include <iostream>
#include <fstream>
using namespace std;

void main()
{
  int i;
  int mas[10] = {1,20,300,4,50,600,7,80,900,10};
```

```
  int vix[10];

  fstream file;
  file.open("ot.txt", ios::out|ios::binary);
  file.write((char*)mas,sizeof(mas));
  file.close();

  file.open("ot.txt", ios::in|ios::binary);
  file.read((char*)vix,sizeof(vix));
  file.close();

  for (i = 0; i < 10; i++)
      cout << vix[i] << " ";
}
```

Для запису масиву до файлу використовується функція **write**, якій треба передати вказівку типу **(char*)** та кількість байтів, які треба записати у бінарному режимі. Отже, вказівку масиву **mas** треба привести до типу вказівки **(char*)**, а у якості кількості байтів передати **sizeof(mas)** (40 байтів). Зчитування відбувається аналогічним чином з використанням функції **read**.

Приклад 2. Записати масив у файл **ot.txt**, прочитати елемент масиву з заданим номером і вивести на екран.

```
#include <iostream>
#include <fstream>
using namespace std;

void main()
{
  int n_el,zn_el;
  int mas[10] = {100,200,3,4,5,6,700,8,9,10};
  fstream file;
  file.open("ot.txt", ios::out|ios::binary);
  file.write((char*)mas,sizeof(mas));
  file.close();

  cout << "Введіть номер елемента -> ";
  cin >> n_el;
  n_el *= sizeof(zn_el);

  file.open("ot.txt", ios::in|ios::binary);
  file.seekg(n_el, ios::beg);
  file.read((char*)&zn_el,sizeof(zn_el));
  cout <<"\n" << zn_el;
  file.close();
}
```

Приклад 3. Записати масив по рядкам у файл **ot.txt**, прочитати і вивести на екран кожний рядок.

```
#include <iostream>
#include <fstream>
using namespace std;

void main()
{
  int i;
  char *mas[3]={"Сидоров","Петров","Іванов"};
  fstream file;
  file.open("ot.txt", ios::out);
  for (i = 0; i < 3; i++)
    file<< mas[i] << "\n";
  file.close();

  char st[100];
  file.open("ot.txt", ios::in);
  file.seekg(0L, ios::beg);
  for (i = 0; i < 3; i++)
  {
    file.getline(st, 100);
    cout << st << "\n";
  }
  file.close();
 }
```

11.3. Файли потокового введення/виведення з використанням структури FILE.

Функції введення/виведення потоком використовують структуру **FILE**. Ця структура і функції потокового введення/виведення описані у файлі <**stdio.h> (<cstdio>)**.

```
typedef struct
{
    short           level;
    unsigned        flags;
    char            fd;
    unsigned char   hold;
    short           bsize;
    unsigned char   *buffer,
    unsigned char   *curp;
    unsigned        istemp;
    short           token;
  } FILE;
```

Деякі з елементів структури FILE можуть бути корисні при програмуванні.

level – порожній або ні буфер введення/виведення;
flags – прапори стану файлу;
fd – номер файлового дескриптору;
hold – непереданий символ;
bsize – розмір буфера (зазвичай 512 байт);
buffer – значення вказівки – адреса початку буфера введення/виведення;
curp – положення поточної вказівки в буфері введення/виведення;
istemp – ознака (прапор) тимчасового файлу;
token – маркер дійсного файлу.

Прапори стану файлу приймають одне з наступних можливих значень, що задаються символьними константами:

_F_RDWR – файл відкритий для читання і запису;
_F_READ – файл відкритий тільки для читання;
_F_WRIT – файл відкритий тільки для запису;
_F_BUF – файл має виділений динамічний буфер даних;
_F_LBUF – порядковий буферизований файл;
_F_ERR – індикатор наявності помилки файлового доступу;
_F_EOF – кінець файлу;
_F_BIN – файл відкритий в двійковому режимі;
_F_IN – здійснюється читання з файлу;
_F_OUT – здійснюється запис у файл;
_F_TERM– файл є терміналом.

Відкриття потоків.

Потік має бути відкритий з використанням функцій **fdopen**, **fopen**, або **freopen** до виконання операцій введення/виведення з цим потоком. Стандартні потоки відкривати не потрібно.

Потік може бути відкритий:

1) тільки для читання;
2) тільки для запису;
3) для читання/запису одночасно;
4) для запису в кінець файлу (режим доповнення).

Крім того, потік може бути відкритий в текстовому або двійковому вигляді в будь-якому з цих випадків. Основною функцією для відкриття файлу потоком є функція fopen: FILE***fopen** (char* pathname, char*type);

Функція відкриває файл, ім'я якого визначене в pathname. Це єдине місце в програмі, де ім'я файлу вказується явно. У pathname може бути вказано також і ім'я пристрою.

Type є символьним рядком, який визначає тип доступу, потрібний для файлу. Type може набувати наступних значень:

"**r**" – Відкрити для читання (файл повинен існувати).

"**w**" – Відкрити порожній файл для запису; якщо файл існує, то його вміст втрачається.

"**a**" – Відкрити для запису в кінець файлу (додавання); файл створюється, якщо він не існує.

"**r+**" – Відкрити для читання і запису (файл повинен існувати).

"**w+**" – Відкрити порожній файл для читання і запису; якщо файл існує, його вміст втрачається.

"**a+**" – Відкрити для читання і додавання; файл створюється, якщо він не існує.

Визначуваний тип має бути сумісний із способом доступу, з яким файл був відкритий.

Коли файл відкритий з "**a**" або "**a+**" типом, дані записуються в кінець файлу. Хоча вказівка файлу може бути переміщений, використовуючи **fseek** або **rewind** функції, але він завжди переміщатиметься назад на кінець файлу при записі даних. Таким чином, існуючі дані не можуть бути затерті.

Коли "**r+**", "**w+**" або "**a+**" типи визначені, читання і запис дозволені (якщо файл відкритий для оновлення). Проте перемикання з читання на запис повинне здійснюватися **fseek** або **rewind** командами. Поточна позиція може бути визначена для **fseek** команди, якщо необхідно.

Для задавання режиму відкриття файлу (текстового/двійкового) використовуються символи **t** і **b**. Ці символи записуються обов'язково після символів типу доступу. Наприклад, для відкриття двійкового файлу в режимі читання/запису можна записати тип у вигляді "**r+b**" або "**rb+**", але не у вигляді "**br+**" або "**b+r**" або "**+rb**".

Функція fopen повертає вказівку на відкритий файл. У разі помилки повертається значення NULL.

Крім того, файл може бути відкритий за допомогою функцій fdopen і freopen. Функція fdopen дозволяє обробляти файл відкритий/створений за допомогою системи введення/виведення нижнього рівня засобами системи введення/виведення верхнього рівня.

Функція **freopen** закриває файл, пов'язаний з потоком **stream** і перепризначає потік **stream** на файл визначений в **pathname**.

FILE* freopen (char* pathname, char* type, FILE* stream);

Ця функція використовується для того, щоб зв'язати стандартні потоки **stdin**, **stdout**, **stderr**, **stdaux** і **stdprn** з файлами що задаються користувачем.

Операції введення-виведення.

1) int fscanf (FILE* stream, char* format [, adress]);

Ця функція виконує форматоване введення з потоку. Вона схожа на функцію **scanf**, за винятком того, що введення проводиться з потоку **stream**. Файл потоку має бути відкритий в текстовому режимі. Функція повертає число успішно введених полів.

2) int fprintf (FILE* stream, char* format [, adress]);

Ця функція виконує форматоване виведення в потік. Вона схожа на функцію **printf**, за винятком того, що виведення проводиться в потік **stream**. Файл потоку має бути відкритий в текстовому режимі.

3) int fgetc (FILE* stream);

Функція вводить символ з потоку **stream**. При успішному введенні функція повертає символ перетворений до типу **int**, без розширення знаку. Якщо виникла помилка, то функція повертає символ EOF. Ця функція успішно працює як в текстовому, так і в двійковому режимі.

4) int fputc (int c, FILE* stream);

Функція виводить символ **c** в потік **stream**. При успішному виведенні функція повертає символ "**c**". Якщо виникла помилка, то функція повертає символ EOF. Ця функція успішно працює як в текстовому, так і в двійковому режимі.

5) int fread (char* buf, unsigned size, unsigned n, FILE* stream);

Функція читає **n** елементів довжини **size** з потоку **stream** і поміщає їх в буфер **buf**. Функція повертає число прочитаних елементів. Якщо виникла помилка, або досягнутий кінець файлу то буде повернено число менше, ніж n. Ніяких перетворень форматів не проводяться. Цю функцію доцільно застосовувати для ефективного обміну даними в двійковому режимі, хоча можливе її застосування і в текстовому режимі.

6) int fwrite (char* buf, unsigned size, unsigned n, FILE* stream);

Функція виводить **n** елементів довжини **size** в потік **stream** з буфера **buf**. Функція повертає число виведених елементів.

Операції переміщення файлового покажчика (позиціонування) і допоміжні функції.

1) void rewind (FILE* stream);

Ця функція переміщує внутрішній покажчик на початок файлу. Вона схожа на описану нижче функцію **fseek**, викликану з параметрами fseek(stream, 0L, 0).

2) int fseek (FILE* stream, long offset, int fromwhere);

Ця функція переміщує внутрішній покажчик файлу в задане положення. Параметр **offset** задає зсув покажчика. Параметр **fromwhere** визначає місце, від якого відлічується зсув **offset** і може мати одне з трьох значень: 0, 1, 2. Ці константи позначені таким чином:

SEEK_SET – 0 – початок файлів;

SEEK_CUR – 1 – позиція поточного покажчика файлу;

SEEK_END – 2 – кінець файлу.

Функція повертає значення 0, якщо покажчик був успішно переміщений, і не 0, у разі помилки.

3) long ftell (FILE* stream);

Ця функція повертає поточне положення внутрішнього покажчика файлу. При помилці функція повертає EOF.

4) int feof (FILE* stream);

Функція виявляє кінець файлу пов'язаного з потоком **stream**. Ця функція повертає ненульове значення (EOF), якщо виявлений кінець файлу, інакше функція повертає 0. Ознака кінця файлу скидається при кожній операції введення.

5) int fflush (FILE* stream);

Функція записує на диск буфер, пов'язаний з потоком **stream**, якщо він був відкритий для виведення. Потік залишається відкритим після роботи цієї функції. Ця функція повертає нульове значення, якщо операція пройшла успішно, інакше функція повертає EOF.

6) int fflushall (void);

Функція записує на диск всі буфери, пов'язані з відкритими потоками виведення та обнуляє буфери потоків введення. Ця функція повертає кількість відкритих потоків (введення і виведення).

Закриття файлу.

1) int fclose (FILE* stream);

Функція закриває потік **stream**, при цьому записується на диск буфер, пов'язаний з потоком **stream**, якщо він був відкритий для виведення. Ця функція повертає нульове значення, якщо операція пройшла успішно, інакше функція повертає EOF. Якщо файл не закритий, то при нормальному закінченні програми він закривається автоматично.

2) int fcloseall (void);

Функція закриває всі відкриті потоки, окрім стандартних: **stdin**, **stdout**, **stderr**, **stdaux**, **stdprn**. Ця функція повертає нульове значення, якщо операція пройшла успішно, інакше функція повертає EOF.

Розглянемо приклад програми роботи з файлами. Записати масив у файл **ot.txt**, прочитати його і вивести на екран. Цей приклад є аналогом прикладу роботи з файлами, де було використано потік **fstream**. У нашому випадку використовується структура FILE. Зробимо це двома способами: з використанням текстових та бінарних файлів.

Приклад 1. Запис та читання масиву, використовуючи текстовий файл.

```
#include <iostream>
using namespace std;

void main()
{
  int i;
  int mas[10] = {1,20,300,4,50,600,7,80,900,10};
  int vix[10];

  FILE* file;

  file = fopen("ot.txt", "w+");
  for (i = 0; i < 10; i++)
    fprintf(file,"%d ",mas[i]);

  fseek(file,0L, SEEK_SET);

  for (i = 0; i < 10; i++)
```

```
  {
    fscanf(file,"%d ",&vix[i]);
    cout << vix[i] << " ";
  }
  fclose(file);
}
```

Приклад 2. Запис та читання масиву, використовуючи бінарний файл.

```
#include <iostream>
using namespace std;

void main()
{
  int i;
  int mas[10] = {1,20,300,4,50,600,7,80,900,10};
  int vix[10];

  FILE* file;

  file = fopen("ot.txt", "w+");

  fwrite(mas,10,sizeof(mas[0]),file);
  fseek(file,0L, SEEK_SET);
  fread(vix,10,sizeof(vix[0]),file);

  for (i = 0; i < 10; i++)
    cout << vix[i] << " ";

  fclose(file);
}
```

Контрольні питання

1. Що таке файл?
2. Які типи файлів ви знаєте?
3. Наведіть функції відкриття та закриття файлів потоку fstream.
4. Перелічите можливі режими доступу до файлів потоку fstream.
5. Що таке довільний доступ у файлах?
6. Опишіть функцію переведення вказівки файлу в довільне місце. Наведіть приклад.
7. Напишіть програму копіювання файлів.
8. Опишіть додаткові функції роботи з файлами – read та write.
9. Наведіть приклади роботи з файлами.
10. Наведіть структуру FILE.

11. Наведіть функції потокового введення даних з використанням структури FILE.

12. Наведіть функції потокового виведення даних з використанням структури FILE.

Завдання

У даній роботі необхідно зробити програму роботи з одновимірними масивами. Бажано використовувати динамічні масиви (виділення пам'яті за допомогою оператора new). Введення/виведення даних організовувати за допомогою файлів даних, використовуючи потоки fstream або FILE.

Варіанти індивідуальних завдань

1. Визначити добуток додатних парних чисел масиву B(n). Якщо таких елементів немає, вивести у файл повідомлення: «Додатних парних елементів в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.
2. Визначити добуток від'ємних чисел масиву A(n), що стоять на парних позиціях. Якщо таких елементів немає, вивести у файл повідомлення: «Від'ємних елементів в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.
3. Визначити суму і кількість непарних чисел масиву M(n). Якщо таких елементів немає, вивести у файл повідомлення: «Непарних елементів в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.
4. У масиві чисел K(n) визначити суму і кількість чисел, кратних 5. Якщо таких елементів немає, вивести у файл повідомлення: «Елементів кратних 5 в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.
5. У масиві чисел X(n) визначити кількість чисел, кратних 2, і чисел, не кратних 3. Якщо таких елементів немає, вивести у файл повідомлення: «Чисел, кратних 2 і чисел не кратних 3 в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.
6. Є масив чисел C(n), визначити кількість і добуток елементів масиву, які знаходяться в діапазоні $0 \le C(i) \le 7$. Якщо таких елементів немає, вивести у файл повідомлення: «Елементів з діапазону [0; 7] в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.
7. У заданому масиві A(n) визначити середнє арифметичне значення чисел, кратних трьом. Якщо таких елементів немає, вивести у файл повідомлення:

«Елементів кратних 3 в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

8. Визначити середнє арифметичне значення елементів масиву F(n), які задовольняють вимозі $-3 \le F(i) \le 5$. Якщо таких елементів немає, вивести у файл повідомлення: «Елементів, що задовольняють вимозі, в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

9. У числовому масиві A(n) визначити мінімальний елемент масиву і його порядковий номер. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

10. Визначити мінімальний елемент числового масиву K(n) і кількість елементів, рівних мінімальному елементу. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

11. Визначити добуток непарних елементів масиву P(n), що стоять на парних позиціях. Якщо таких елементів немає, у файл вивести повідомлення: «Непарних елементів, що стоять на парних позиціях в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

12. Визначити середнє арифметичне значення елементів масиву B(n), кратних восьми. Якщо таких елементів немає, у файл вивести повідомлення: «Елементів кратних 8 в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

13. У числовому масиві Z(n) серед додатних елементів визначити максимальний елемент масиву. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

14. Визначити кількість від'ємних елементів в масиві A(n) і на їх місце записати нулі. Якщо таких елементів немає, у файл вивести повідомлення: «Від'ємних елементів в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

15. Визначити добуток додатних елементів масиву і їх кількість. За відсутності додатних чисел вивести у файл повідомлення «Додатних чисел в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

16. Обчислити кількість елементів цілочисельного масиву V(n), кратних семи. За відсутності таких елементів вивести у файл повідомлення «Елементів, кратних 7, немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

17. Для числового масиву Z(n) визначити середнє арифметичне значення мінімального і максимального елементів. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

18. Для заданого масиву A(n) обчислити суму і кількість елементів, що задовольняють умові $2 \le A(i) \le 10$. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

19. Визначити суму і добуток додатних чисел, що стоять на парних позиціях в масиві B(n). Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

20. У масиві C(n) визначити середнє арифметичне значення додатних елементів і середнє арифметичне значення від'ємних елементів. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

21. У масиві B(n) визначити окремо кількість від'ємних чисел, кількість додатних чисел і чисел, рівних нулю. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

22. Визначити суму додатних парних чисел масиву B(n). У разі відсутності додатних парних чисел вивести у файл повідомлення «Додатних парних чисел в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

23. Визначити середнє арифметичне значення чисел масиву A(n), що стоять на парних позиціях, і середнє арифметичне значення чисел масиву A(n), що стоять на непарних позиціях. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

24. Визначити суму додатних елементів масиву B(n), що стоять на позиціях, кратних трьом. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

25. Заданий масив цілих чисел C(n). Обчислити добуток максимального і мінімального елементів. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

26. У масиві чисел B(n) всі по додатні елементи замінити на нульові значення і визначити середнє арифметичне значення від'ємних елементів масиву. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

27. У масиві чисел C(n) числа, що стоять на парних позиціях, замінити на нулі і визначити середнє значення чисел, що стоять на непарних позиціях. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

28. У числовому масиві A(n) визначити індекси мінімального і максимального елементів. Визначити різницю між максимальним та

мінімальним елементами. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

29. Для заданого цілочисельного масиву M(n) обчислити добуток і кількість від'ємних непарних чисел, розташованих на парних позиціях. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

30. Заданий цілочисельний масив A(n). Обчислити суму і кількість чисел, що діляться без остачі на 5. Якщо таких чисел немає, у файл необхідно вивести повідомлення «Чисел, що діляться без остачі на 5, в масиві немає». Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

31. У заданому масиві A(n), розділити всі елементи масиву A(n) на п'ятий елемент масиву A(n). У перетвореному масиві обчислити суму елементів, що стоять на непарних позиціях. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

32. Заданий масив чисел X(n). Обчислити кількість від'ємних чисел і вивести у файл індекс першого від'ємного елементу і індекс останнього від'ємного елементу. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

33. Є цілочисельний масив B(n). Перетворити заданий масив шляхом множення кожного елементу масиву на максимальний елемент цього масиву. Вивести у файл перетворений масив і максимальний елемент цього масиву. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

34. Заданий масив чисел A(n). Перетворити заданий масив шляхом ділення кожного парного елементу масиву на 2. Обчислити середнє арифметичне значення елементів перетвореного масиву. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

35. У масиві C(n) визначити кількість елементів рівних нулю і на їх місце записати число 5. Обчислити середнє арифметичне значення елементів, рівних 5. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

РОЗДІЛ 12. СТРУКТУРИ

12.1. Загальна характеристика структури

Структури дають можливість групувати дані і працювати з цими даними як із єдиним цілим.

Структура являє собою множину, що складається з декількох значень, кожне з який може мати свій тип даних. Елементи структур можуть мати різні типи даних.

Для визначення структури використовується синтаксис:

struct [тег структури]
{
визначення члена структури;
визначення члена структури; } поля структури
} [одна або декілька змінних-структур];

тег-ім’я структури.

Приклад структури з інформацією про компакт-диски:

```
struct cd_info
{
 char title [25];        //назва
 char artist [28];       //виконавець
 int num_song;           //число пісень
 float price;            //вартість
 char date_bought [8];   //дата покупки
} cd1, cd2, cd3;
```

Тут описані три структурні змінні: **cd1, cd2, cd3**;

Існує два засоби ініціалізації структур:

- при оголошенні;
- окремо в тілі програми.

Приклад 1. Ініціалізація при оголошенні

```
struct cd_info
{
 char title[25];
 char artist[28];
 int num_song;
 float price;
 char date_bought[8];
} cd1=
{
"Red Moon Men",
"Пугачова", 12, 20.55,
```

```
"02/13/98"
};
```

Приклад 2. Присвоювання в тілі програми.

```
#include <iostream>
using namespace std;

  struct cd_info
  {
    char title [25];
    char artist [28];
    int num_song;
    float price;
    char date_bought [8];
  };

void main()
{
  cd_info cd1;
  strcpy(cd1.title, "Red Moon Men");
  cd1.num_song = 12;
  cd1.price = 11.95f;
  strcpy(cd1.artist, "Пугачова");
  cout << "Title: " << cd1.title << "\n";
  cout << "Artist: " << cd1.artist << "\n";
  cout << "Number of songs: " << cd1.num_song<< "\n";
  cout << "Price: " << cd1.price << "\n";
}
```

Як очевидно з прикладу до полів структури можна звертатися, вказуючи ім’я змінної-структури, ім’я поля та розділяючи ці імена точкою.

У загальному вигляді:

```
struct bb
{
  int field_1;
  char field_2;
} c={10,'A'}, a={20,'B'};
int value1 = c.field_1; //10
char value2 = a.field_2; //B
```

Якщо змінна **ptr** визначена як вказівка на структуру, то для доступу до полів структури використовується операція "–>". Перед цим необхідно створити об’єкт структури, використовуючи оператор **new**, а в кінці видалити його за допомогою **delete**.

Наприклад:

```
bb *ptr = new bb;
ptr->field_1 = 10;
ptr->field_2 = 'A';
.........................
delete bb;
```

Оператор `ptr->field_1` еквівалентний виразу `(*ptr).field_1`.

Приведемо приклад роботи з комплексними числами:

```
#define COMPLEX struct complex_type

COMPLEX
{
float real;
float image;
};
```

Операція додавання C1 і C2 для одержання C3.

```
#define COMPLEX struct complex_type

COMPLEX
{
  float real;
  float image;
};

COMPLEX C1, C2, C3;
C1.real = 1;
C1.image = 2.5f;
C2.real = 2;
C2.image = 3;
C3.real = C1.real + C2.real; //3
C3.image = C1.image + C2.image; //5.5
```

або

```
COMPLEX* C1, *C2, *C3;
C1 = new COMPLEX;
C2 = new COMPLEX;
C3 = new COMPLEX;
C1->real = 1;
C1->image = 2.5f;
C2->real = 2;
C2->image = 3;
C3->real = C1->real + C2->real; //3
C3->image = C1->image + C2->image; //5.5
.................................
```

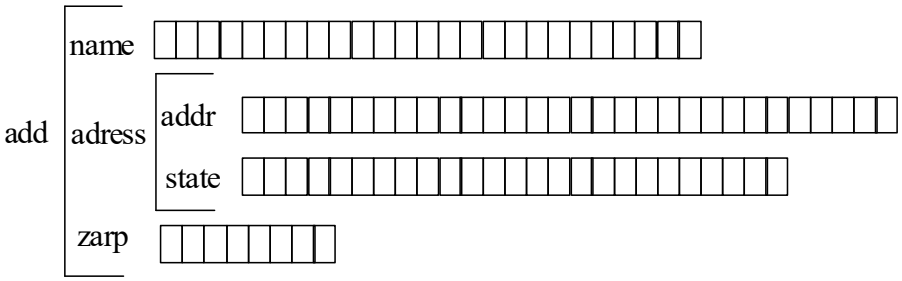
```
delete C1;
delete C2;
delete C3;
```

Структури можуть бути вкладеними, тобто одним із полів структури може бути інша структура.

Приклад:

```
#include <iostream>
using namespace std;

struct address_info
{
  char addr[25]; //Інформація про адресу
  char state[20];
};

struct add
{
  char name[25];
  address_info *address;
  double zarp;
};

void main()
{
  add t;
  t.address = new address_info;
  cin >> t.zarp;
  cin >>t.name;
  cin >>t.address->state;
  cin >>t.address->addr;

  delete t.address;
}
```

Вкладена структура має вигляд:



Резюме:

Структури дозволяють групувати дані більш гнучким чином, ніж масиви. Вони можуть містити елементи різноманітних типів даних. Ініціалізувати структури можна двома засобами: під час оголошення, або за допомогою операторів «точка»(.) або "–>" у тілі програми. Структури аналогічні записам бази даних, а їхні елементи – полям записів.

12.2. Масиви структур

Припустимо, що ви працюєте в компанії замовлення поштою, яка продає дисководи. Перед вами стоїть задача написати програму для обліку 25 видів пристроїв. По дисководу є наступна інформація:

- місткість пристрою;
- час доступу в мілісекундах;
- код постачальника;
- ціна.

Дані зручно розмістити в масиві структур

```
struct inf_d
{
long int ob;
int vr;
int kod;
double cena;
} drive[25];
```

або окремим рядком:

```
inf_d drive[25];
```

Структури зручно записувати на диск

Приклад:

```
#include <iostream>
using namespace std;

#include <conio.h>

struct inf_d //Глобальне визначення
{
  long int ob;
  int vr;
  int kod;
  double cena;
};
```

```
inf_d v_dan()
{
  inf_d disk;
  cin >> disk.ob;
  cin >> disk.vr;
  cin >> disk.kod;
  cin >> disk.cena;
  return disk;
}

void menu()
{
  system("cls"); // Очищення екрану
  cout << "1. введення даних про дисковод\n";
  cout << "2. Відображення даних\n";
  cout << "3. Вихід\n";
}

void pros(inf_d disk[25], int num)
{
  for(int ctr = 0; ctr < num; ctr++)
  {
    cout <<"\n" << disk[ctr].ob ;
    cout <<"\n" << disk[ctr].vr;
    cout <<"\n" << disk[ctr].kod;
    cout <<"\n" << disk[ctr].cena<<"\n";
  }
    getch();
}

void main ()
{
  inf_d disk[25];
  int ans;
  int num = 0;
  do
  {
    do
    {
       menu();
       cin >> ans;
    } while ((ans < 1) || (ans>3));

    switch(ans)
    {
       case 1:
```

```
          disk[num] = v_dan();
          num++;
          break;
        case 2:
          pros(disk,num);
          break;
        default:
          break;
    }
  }
  while(ans!=3);
}
```

У даному прикладі для очищення екрану використовується системна функція **system** з параметром **"cls"** (**system("cls")**). Для очікування введення клавіші <Enter> у режимі перегляду даних використовується функція **getch**. Ця функція знаходиться у заголовному файлі **<conio.h>**.

12.3. Використання масивів, як елементів структур

Дано структури платіжних відомостей:

```
struct pl
{
  char name[25];  //ім'я службовця
  char otd[10];   //назва відділу
  double zar;     //зарплата
} mas_pl[100];
```

Припустимо необхідно записати в символьну змінну ініціал ім'я 25-го службовця:

```
char iniz;
iniz = pl[24].name[0];
```

Якщо необхідно змінити ім'я службовця, тоді:

```
strcpy(pl[24].name, "Іван Петрович");
```

Приклад. Книгарня створює каталог книг.

```
#include <iostream>
using namespace std;

#include <conio.h>

struct inven
{
```

```
  char title[25]; //заголовок
  char pub[19];   //Дата публікації
  char avt[20];   //Ім'я автора
  float cena;
};

void main()
{
  inven kn[100];
  int total = 0;
  char ans;
  do //введення даних
  {
    cin>>kn[total].title;
    cin>>kn[total].pub;
    cin>>kn[total].avt;
    cin >> kn[total].cena;
 cout << "\nБудете вводити? (Y/N) ";
    ans = getch();
ans = toupper(ans);
    if(ans == 'Y')
    {
      total++;
      cout<<"Y\n";
      continue;
    }
  }
  while(ans != 'N');
}
```

Приклад запису на диск масиву структур:

Така функція може бути частиною великої програми, що одержує дані досліджень у масив структур від користувача. Дана функція приймає ім'я масиву та число елементів. Функція **write** записує масив на диск.

```
#include <iostream>
using namespace std;

ofstream fp;

struct inv
{
  char title[25]; //заголовок
  char pub[19];   //Дата публікації
  char avt[20];   //Ім'я автора
  float cena;
};
```

```
void write (struct inv m[], int kol)
{
  fp.write(m, kol*sizeof(struct inv));
}
```

Якщо масив із даними досліджень має 1000 елементів, дана функція буде все одно записувати їх за один раз. Крім того, можна використовувати функцію **read()**, щоб зчитувати вміст масиву за один виклик функції.

Слід сказати, що при роботі в консольному режимі виникають проблеми з кодуванням символів. З цією метою у приведеному прикладі запропоновано функцію recode(), яка переводить символьні данні з таблиці кодування Windows у DOS. Крім того, за допомогою функції IsNumeric() передбачено контроль введення числових даних (false(0) – контроль цілих значень, true(1)–дробових). Якщо введено число – функція повертає значення true, якщо ні – false.

<u>УВАГА!!!</u> Необхідно звернути увагу на наступні моменти програми:

1. У програму додані дві функції, які стосуються виведення даних: функція очищення екрану cls та функція переходу на будь-який рядок GotoXY. Для роботи з консольними функціями необхідно включити хайдер <windows.h> до програми. Приведемо текст цих двох функцій:

```
void cls()
{
 system("cls");
}

void GotoXY(int X, int Y)
{
 COORD coord = { X, Y };
 SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),coord);
}
```

2. З метою коректного введення та контролю даних використовується функція gets. Функцію потокового введення cin використовувати недоцільно, так як немає контролю натискання клавіші <Enter>.

3. Робота зі структурами є універсальною, передбачено додавання записів до файлу (флаг ios:app). Для підрахунку кількості структур на диску приведемо наступні рядки програми:

```
.....
  pp.seekg(0L,ios::end);
  kol_st = pp.tellg()/sizeof(inf_d);
.....
```

Спочатку встановлюється вказівка у кінець файлу, а потім ділимо поточну позицію файлу (функція tellg()) на кількість байтів нашої структури inf_d. Таким чином отримуємо кількість структур на диску (змінна kol_st).

4. Після переведення вказівки на кінець файлу (функція eof()) виникає біт помилки ios::eofbit, що дорівнює 1. Для того щоб продовжити роботу з файлом потоку, необхідно скинути цей біт. Робиться це за допомогою функції clear(), яка скидує всі біти помилок файлового потоку.

Приведемо текст програми роботи зі структурами з використанням інформації по дискам.

```
//Програма роботи зі структурами
//з використанням диску

#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

#include <conio.h>

#include <windows.h>

fstream pp;

struct inf_d  //Визначення структури
{
 long int ob; //число оборотів
 int vr;      //час доступу до диску
 long int kod;//код виробу
 double cena; //ціна виробу
 char   firma[20]; //назва фірми
};

int  menu(void);  //Оголошення функцій: меню,
void v_dan(void);  //перегляду,
void readst(int nom = 0); //читання структури
int  vib_struct(void); // вибір елемента структури
//   переведення символів із кодировки Windows у
     кодировку DOS
char* recode(char *TXT);
// перевірка символів на числа
bool IsNumeric(char *str, bool dr);

//Функція очищення екрану
void cls()
{
 system("cls");
}
```

```
//Функція переходу у точку з координатами X,Y для виведення даних
void GotoXY(int X, int Y)
{
 COORD coord = { X, Y };
 SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),coord);
}

// Головна функція програми
void main ()
{

    int nom_menu;
    pp.open("Ot.txt",(ios::in)|(ios::out)
                          |(ios::app)|(ios::binary));

    while(1)
    {
        cls();
        nom_menu = menu();
        if (nom_menu == 1) v_dan();
        if (nom_menu == 2) readst();
        if (nom_menu == 3) readst(vib_struct());
        if (nom_menu == 4) break;
    }

   pp.close();  //Закриття файла
}
//**************МЕНЮ*********************

//Визначення функції меню
int menu()
{
  char s[50];
  int nom_menu;
  long kol_st;
  pp.seekg(0L,ios::end);
  kol_st = pp.tellg()/sizeof(inf_d);

       while(1)
    {
       nom_menu = 0;

  cls();
  GotoXY(27,4);
  cout<<recode("Всего структур ")<<kol_st;
  GotoXY(20,6);
```

```
  cout<<
   "====================================";
  GotoXY(20,7);
  cout<<recode("| 1 - Ввод данных по дисководу     |");
  GotoXY(20,8);
  cout<<recode("| 2 - Отображение всех данных      |");
  GotoXY(20,9);
  cout<<recode("| 3 - Просмотр заданной структуры  |");
  GotoXY(20,10);
  cout<<recode("| 4 - Выход                        |");
  GotoXY(20,11);
  cout<<
   "====================================\n";


   GotoXY(27,13);
   cout << recode("Введите пункт меню ");
   gets(s);

if (IsNumeric(s, 0))nom_menu = atoi(s);
if ((nom_menu < 1 )||(nom_menu > 4))
 {
 GotoXY(18,18);
 cout<<       "----------------------------------------";
 GotoXY(18,19);
 cout<<recode("|  Ошибка! Пункт меню должен быть 1 - 4 |");
 GotoXY(18,20);
 cout<<recode("|          Нажмите любую клавишу        |");
 GotoXY(18,21);
 cout<<       "----------------------------------------";
 getch();
 GotoXY (44,13);
 cout << "                ";
 continue;
 }
  return  nom_menu;
 }
}
// Функція вибору номера структури
int vib_struct()
{
  char s[50];
  int nom_st;
  long kol_st;
  pp.seekg(0L,ios::end);
  kol_st = pp.tellg()/sizeof(inf_d);
```

```
  if(!kol_st)return 0;

  while(1)
  {
  nom_st = 0;
  cls();

  GotoXY(1,2);
  cout<<recode("Введите номер структуры -> ");
  gets(s);
  if(IsNumeric(s, 0))nom_st=atoi(s);
  if(nom_st<1) continue;
  if(nom_st<=kol_st)break;

GotoXY(18,10);
cout<<        "-----------------------------------------";
GotoXY(18,11);
cout<<recode("|              Внимание!!!                |");

GotoXY(18,12);
cout<< recode("|       В файле нет")<<setw(3)<<nom_st;
cout<<recode("-й структуры       |");

GotoXY(18,13);
cout<< recode("|   Файл содержит только")<<setw(3)<<kol_st;
cout<<recode(" структур(ы)  |");

GotoXY(18,14);
cout<<recode("|         Нажмите любую клавишу           |");
GotoXY(18,15);
cout<<        "-----------------------------------------";

  getch();
 }

  return nom_st;
}

//*****************ВВЕДЕННЯ**********************

// Визначення функції введення даних
void v_dan()
{

  inf_d dis;
```

```
  int i;
  char s[50];
  cls();

  for(i=0;i<4;i++)
  while(1)
  {
  GotoXY(1,2+2*i);
  if(i==0)cout<<recode("Введите число оборотов диска ->");
  if(i==1)cout<<recode("Введите время доступа к диску ->");
  if(i==2)cout<<recode("Введите код товара ->");
  if(i==3)cout<<recode("Введите цену дисковода ->");

  GotoXY(35,2+2*i);
  gets(s);
  if(IsNumeric(s,(i==3)?1:0))
  {
   if(i==0)dis.ob=atoi(s);
   if(i==1)dis.vr=atoi(s);
   if(i==2)dis.kod=atoi(s);
   if(i==3)dis.cena=atof(s);
   break;
  }

  GotoXY(35,2+2*i);
  cout<<"                  ";
  }

  GotoXY(1,10);
  cout << recode("Введите название фирмы ->");

  GotoXY(35,10);
  gets(dis.firma);

//переведення курсору на кінець файла
  pp.seekg(0L,ios::end);
  pp.write((char*)&dis,sizeof(dis));//Запис на диск
}

//****************ЧИТАННЯ*********************
//Функція читання даних з диску
void readst(int nom_st)
{
    inf_d od;
    long kol_st, ch_b = 0;
    cls();
```

```
    pp.seekg(ch_b,ios::end);

    kol_st = pp.tellg()/sizeof(inf_d);

if(!kol_st)
{
 GotoXY(18,10);
 cout<<        "----------------------------------------";
 GotoXY(18,11);
 cout<<recode("|  Внимание! В файле нет информации     |");
 GotoXY(18,12);
 cout<<recode("|    Для ввода используйте пункт 1 меню |");
 GotoXY(18,13);
 cout<<recode("|         Нажмите любую клавишу         |");
 GotoXY(18,14);
 cout<<        "----------------------------------------";
 getch();
 return;
}


    if(nom_st > 0)ch_b = sizeof(od)*(nom_st-1);
    pp.seekg(ch_b,ios::beg);//Відкриття файла
    //Читання файла з диска

    cout<<recode("\n   Данные по дискам:\n");

    while(1)
    {
    pp.read((char *)&od, sizeof(od));
     // Очистка біту помилки (clear()) та вихід з циклу
    if(pp.eof()){pp.clear();break;}
    cout<<"\n";
    cout<<recode("   обороты-")<<setw(3)<<od.ob;
    cout<<recode("   время-")<< setw(5)<<od.vr;
    cout<<recode("   код-") << setw(3)<<od.kod;
    cout<<recode("   цена-")<< setw(7)<<od.cena;
    cout<<recode("   фирма- ")<< od.firma;
    if(nom_st)break;
    }
   getch();
}
```

```
// Функція переведення кодировки із Windows у DOS
char* recode(char *TXT)
{

static char buf[257];
strcpy(buf,TXT);

unsigned int i,kol;
kol = strlen(buf);

for(i=0;i<kol;i++)
       {
        if (buf[i]>='A'&&buf[i]<='п')buf[i]-=64;
        if (buf[i]>='р'&&buf[i]<='я')buf[i]-=16;
       }

return buf;
}

// Функція перевірки символів на числа
// Вхід:
// str - строка символів;
// dr = false (0) - перевірка на ціле число;
// dr = true (1) - перевірка на дробове число;
// Повертання:
// true - перевірка пройшла успішно
// false - введено не числове значення, або число не
   задовільняє умовам.

bool
IsNumeric(char *str, bool dr)
{
  int i;
  if (!strlen(str) || str[0] == '.')return false ;
    for (i = 0; i < (int)strlen(str); i++ )
    {
     if (i == 0 && (str[0] == '+' ||  str[0] == '-'))
continue;
     if ((str[i] >= '0' && str[i] <='9') ||
         str[i]=='.')  continue;
     return false;
    }
     int pos = 0;
     for (i = 0; i < (int)strlen(str);i++)
     if (str[i]=='.')pos++;
     if ( (!dr && pos) || (pos > 1)) return false ;
     return true;
}
```

Контрольні питання

1. Що таке структура? Опис структури.
2. Наведіть приклади ініціалізації структури.
3. Як записати структуру до файлу?
4. Як зчитати структуру з файлу?
5. Розкрийте сутність вкладених структур.
6. Розкрийте сутність масивів структур.
7. Наведіть приклад запису на диск масиву структур.
8. Наведіть приклад зчитування з диску масиву структур.
9. Розкрийте сутність функції перекодування рядків із Windows у DOS – recode ().
10. Розкрийте сутність функції контролю числових даних IsNumeric().

Завдання

На основі приведеного прикладу розробити самостійно програму роботи зі структурами, в якій буде відбуватись комплексна робота з даними.

РОЗДІЛ 13. ОБ'ЄДНАННЯ ТА ІНШІ ТИПИ ДАНИХ. ОБРОБКА ВИКЛЮЧНИХ СИТУАЦІЙ

13.1. Об'єднання

Відмінність об'єднань (union) від структур (struct) полягає в тому, що всі дані, які є членами об'єднання, знаходяться в одній і тій же області пам'яті.

Розглянемо приклад використання об'єднання для побайтового виведення значення типу **double** у двійковому представленні.

```
#include <iostream>
using namespace std;

union bits
{
 double d;
 unsigned char c[sizeof(double)];
};

void main()
{
 bits ob1;
 ob1.d = 15.75;
 int i,j;
 for(i=0;i<sizeof(double);i++)
  {
   cout<<"\nБайт "<<i<<" :";
   for(j=128;j;j>>=1)
       if(ob1.c[i] & j)cout<<"1"; else cout<< "0";
  }
}
```

Результат виконання програми приймає наступний вигляд:

```
Байт 0 :00000000
Байт 1 :00000000
Байт 2 :00000000
Байт 3 :00000000
Байт 4 :00000000
Байт 5 :10000000
Байт 6 :00101111
Байт 7 :01000000
```

13.2. Перелічені типи даних (enum)

При написанні програм часто виникає потреба визначити декілька іменованих констант, для яких потрібно, щоб всі вони мали різні значення. Для цього зручно скористатися переліченим типом даних, всі можливі значення якого задаються списком цілочисельних констант. Формат:

```
enum [ ім'я_типу ] { список_констант };
```

Ім'я типу задається в тому випадку, якщо в програмі потрібно визначати змінні цього типу. Компілятор забезпечує, щоб ці змінні приймали значення тільки із списку констант. Константи мають бути цілочисельними і можуть ініціалізуватися звичайним способом. За відсутності ініціалізатора перша константа обнуляється, а кожній наступній привласнюється на одиницю більше значення, чим попередній. Приведемо приклад:

```
#include <iostream>
using namespace std;

void main()
{
// enum svet{red,amber,green}; // варіант 1
   enum svet{red=5,amber=9,green=8}; //варіант 2

 svet p1 = amber;
 switch(p1)
{
 case red:
          cout<<"red\n";break;
 case amber:
          cout<<"amber\n";break;
 case green:
          cout<<"amber\n";break;
}
}
```

У варіанті 1 тип даних **svet** є цілим типом даних (**int**) в якому мають місце три перелічені значення. Значення константи **red** відповідатиме 0, **amber** – 1, **green** – 2. Можна явно указати кожній константі своє значення (варіант 2).

Слід зауважити, що тип даних **enum** еквівалентний типу даних **int** і можна виконувати всілякі операції між даними типами. Але, якщо до типу даних **enum** додається змінна типу **int** – то отриманий результат слід привести до потрібного типу даних, як це зроблено в рядку:

```
svet p1 = (svet)(amber+1);
```

Результатом буде: p1 = 10.

Приведемо ще один засіб застосування типу enum.

```
enum {red=2,amber,green=5};
```

Константі **amber** привласнюється значення 3. Імена перелічених констант мають бути унікальними, а значення можуть збігатися. Перевага застосування перелічення перед описом іменованих констант і директивою #define полягає в тому, що зв'язані константи наочніші; крім того, компілятор при ініціалізації констант може виконувати перевірку типів.

Універсальність типу даних **enum** полягає в тому, що можна користуватись як зарезервованими константами цього типу, так і довільними значеннями типу даних **int**, що дає програмісту більш широкі можливості.

13.3. Бітові поля

Цілочисельні компоненти можуть бути поміщені у невеликий об'єм пам'яті з використанням бітових полів, тобто інформація записується в окремі біти. Бітові поля розташовуються від менших номерів до більших.

Розглянемо приклад роботи з бітовими полями.

```
#include <iostream>
using namespace std;

struct bit_type
{
  int          i:2;
  unsigned int j:6;
  int           :3;
  unsigned int k:3;
  unsigned int l:2;
} bit;

void main()
{
 bit.i = 2;
 bit.j = 9;
 bit.k = 7;
 bit.l = 2;
 printf("bit=%u",bit);
}
```

Перше поле *i* має розмір 2 біти, друге поле *j* займає 6 бітів. Наступне поле не має назви. Поле *k* обмежене трьома бітами, *l* – обмежено двома бітами.

Наведемо розподіл пам'яті під змінну **bit**.

15 14	13 12 11	10 9 8	7 6 5 4 3 2	1 0
l	*k*	-----	*j*	*i*

Бітове представлення змінної **bit** прийме наступний вигляд.

1 0	1 1 1	0 0 0	0 0 1 0 0 1	1 0

13.4. Обробка виключних ситуацій

C++ забезпечує вбудований механізм обробки помилок, який має назву обробка виключних ситуацій (exception handling). За допомогою обробки виключних ситуацій можна спростити управління на реакцію помилки під час виконання програми. Обробка виключних ситуацій в C++ організовується за допомогою трьох ключових слів: **try**, **catch**, **throw**. Програмний код, у якому треба відслідкувати помилку повинен знаходитись усередині блоку **try**. Якщо у цьому блоці буде знайдена помилка, управління автоматично передається блоку **catch** для її аналізу та обробки. За допомогою інструкції **throw** відбувається генерування помилки, при цьому управління передається блоку **catch** – для обробки даної помилки.

Нижче представлена основна форма інструкцій **try** і **catch**.

```
try
{
        // блок у якому можуть відбуватись помилки
}
catch(type1 arg)
{
        //блок перехвату виключної ситуації
}
catch(type2 arg)
{
        //блок перехвату виключної ситуації
}
catch(typeN arg)
{
        //блок перехвату виключної ситуації
}
```

Блок **try** повинен містити ту частину програми в якій необхідно відстежувати помилки. Це можуть бути як декілька операторів всередині однієї функції, так і всі оператори всередині функції **main()**.

З блоком **try** може бути пов’язано більше однієї інструкції **catch**. Яка саме інструкція **catch** використовується, залежить від типу виключної ситуації. Тобто, якщо тип даних, вказаний в інструкції **catch**, відповідає типу виключної ситуації, виконується дана інструкція **catch**. Можна перехоплювати довільні типи даних, включаючи і типи створених класів.

За допомогою інструкції **throw** можна генерувати виключну ситуацію:

throw [вираз];

Оператор **throw** повинен виконуватися або всередині блоку **try**, або в будь-якій функції, яку цей блок викликає. Якщо ви генеруєте виключну ситуацію, для якої немає відповідної функції **catch**, може відбутися ненормальне завершення програми.

У наступному прикладі показано, як в C++ функціонує система обробки виключних ситуацій.

```
#include <iostream>
using namespace std;

void main()
{
 cout<<"початок\n";
 try
 {
  cout<<"Всередині блоку try\n";
  throw 13; //генерація помилки
  cout<<"Цей рядок не буде виведено";
 }
 catch(int i)
 {
     cout<<"Перехоплена помилка номер: "<<i<<"\n";
 }
 cout<<"кінець";
}
```

Результат виконання програми виглядатиме таким чином:

```
початок
Всередині блоку try
Перехоплена помилка номер: 13
кінець
```

У прикладі є блок **try**, який містить три оператора, та блок **catch(int i)**, який оброблює виключну ситуацію цілого типу. Всередині блоку **try** будуть виконані тільки два з трьох операторів – **cout** і **throw**. Після того, як виключна ситуація збуджена, управління передається виразу **catch** та виконання блоку **try** завершується. Таким чином блок **catch** викликається неявно, управління виконанням програми просто передається йому. Отже, наступний за оператором **throw** оператор **cout** не буде виконаний.

Виключна ситуація може бути згенерована оператором **throw**, що не входить в блок **try**, якщо сам цей оператор входить у функцію, яка викликається з блоку **try**. Приведемо приклад.

```
#include <iostream>
using namespace std;

void Test(int t)
{
 cout<<"Всередині функції t = "<<t<<"\n";
 if(t) throw t;
}
```

```
void main()
{
 cout<<"початок\n";
 try
 {
  cout<<"Всередині блоку try\n";
  Test(0);
  Test(1);
  Test(2);
 }
 catch(int i)
 {
     cout<<"Перехоплена помилка номер: "<<i<<"\n";
 }
 cout<<"кінець";
}
```

Результат виконання програми:

```
початок
Всередині блоку try
Всередині функції t = 0
Всередині функції t = 1
Перехоплена помилка номер: 1
кінець
```

Блок **try** можна розташовувати всередині функції. В цьому випадку при кожному вході у функцію обробник виключної ситуації встановлюється знову. Приведемо наступний приклад.

```
#include <iostream>
using namespace std;

void Test(int t)
{
 try
 {
   if(t) throw t;
 }
 catch(int i)
 {
     cout<<"Перехоплена помилка номер: "<<i<<"\n";
 }
}

void main()
{
 cout<<"початок\n";
```

```
 Test(0);
 Test(1);
 Test(2);

 cout<<"кінець";
}
```

Результат виконання програми:

```
початок
Перехоплена помилка номер: 1
Перехоплена помилка номер: 2
кінець
```

У даному прикладі оброблено три виключні ситуації. Після виклику кожної виключної ситуації функція повертає своє значення.

Як було розглянуто раніше, з блоком **try** можна зв'язати більш ніж один блок **catch**. При цьому кожна інструкція **catch** призначена для перехоплення свого типу виключної ситуації. В тому випадку якщо виключна ситуація не включає відповідний тип блоку **catch**, можна скористатися блоком **catch(…)** в який потрапляють виключення, згенеровані оператором **throw**, які не потрапили в типізовані блоки **catch**.

У наступному прикладі перехоплюються три виключні ситуації: одна для цілих чисел, одна для рядка та одна для будь-яких інших типів даних.

```
#include <iostream>
using namespace std;

void Test(int t)
{
 try
 {
   if(t==0) throw t;
   else if(t==1)throw "Значення рівне 1";
   else throw 25.3;
 }
 catch(int i)
 {
     cout<<"Перехоплена помилка номер: "<<i<<"\n";
 }
 catch(char*s)
 {
     cout<<"Перехоплено рядок: "<<s<<"\n";
 }
 catch(...)
 {
```

```
        cout<<"Перехоплено виключення(крім цілого типу
та рядку)\n";
     }

    }

    void main()
    {
     cout<<"початок\n";

     Test(0);
     Test(1);
     Test(2);

     cout<<"кінець";
    }
```

Після виконання програми побачимо наступний результат:

```
початок
Перехоплена помилка номер: 0
Перехоплено рядок: Значення рівне 1
Перехоплено виключення(крім цілого типу та рядку)
кінець
```

Контрольні питання

1. Що таке об’єднання?
2. В чому відмінність об’єднань від структур?
3. Розкрийте сутність переліченого типу даних (enum).
4. Наведіть приклад роботи з переліченим типом даних enum.
5. Яким чином можна використовувати бітові поля?
6. Наведіть приклад використання бітових полів.
7. Розкрийте сутність та призначення виключних ситуацій.
8. Яким чином можна згенерувати власну помилку?

РОЗДІЛ 14. ДИНАМІЧНІ СТРУКТУРИ ДАНИХ

Будь-яка програма призначена для обробки даних, від способу організації яких залежать алгоритми роботи, тому вибір структур даних повинен робитись перед створенням алгоритмів. Найчастіше в програмах використовуються масиви, структури і їх поєднання, наприклад, масиви структур, полями яких є масиви та структури.

Пам'ять під дані виділяється або на етапі компіляції (в цьому випадку необхідний об'єм має бути відомий до початку виконання програми, тобто заданий у вигляді константи), або під час виконання програми за допомогою операції new. У обох випадках виділяється безперервна ділянка пам'яті.

Якщо до початку роботи з даними неможливо визначити, скільки пам'яті буде потрібно для їх зберігання, пам'ять виділяється в міру необхідності окремими блоками, пов'язаними один з одним за допомогою вказівок. Такий спосіб організації даних називається динамічними структурами даних, оскільки їх розмір змінюється під час виконання програми. З динамічних структур в програмах найчастіше використовуються стеки, черги, лінійні списки. Вони розрізняються способами зв'язку окремих елементів і допустимими операціями.

Динамічні структури широко застосовують і для ефективнішої роботи з даними, розмір яких відомий, особливо для вирішення задач сортування, оскільки впорядковування динамічних структур не вимагає перестановки елементів, а зводиться до зміни вказівок на ці елементи. Наприклад, якщо в процесі виконання програми потрібно багато разів упорядковувати великий масив даних, має сенс організувати його у вигляді лінійного списку.

Елемент будь-якої динамічної структури даних є структурою (struct), що містить принаймні два поля: для зберігання даних і для вказівки. Полів даних та вказівок може бути декілька. Опис простого елементу виглядає таким чином:

```
struct Node
{
Data d; // тип даних Data має бути визначений раніше
Node *p;
};
```

Розглянемо реалізацію основних операцій з динамічними структурами даних (стек, черга, лінійний список) [10, с.114].

14.1. Стек

Стек реалізує принцип обслуговування LIFO (last in – first out, останнім прийшов, – першим пішов). Стек можна представити як стопку книг, які складаються одна на одну. Так першою буде взята остання книга в стопці.

Нижче приведена програма, яка формує стек з п'яти цілих чисел (1, 2, 3, 4, 5) і виводить його на екран. Функція поміщення елементу в стек називається push, а вибірки – pop. Вказівка для роботи із стеком (top) завжди посилається на його вершину.

```
#include <iostream>
using namespace std;

struct Node
{
 int d;
 Node *p;
};

Node * first(int d);
void push(Node **top, int d);
int pop(Node **top);

void main()
{
 Node* top = first(1);
 for (int i = 2; i<6; i++) push(&top, i);
 while (top)
   cout << pop(&top) << " ";
}

// Початкове формування стеку
Node * first(int d)
{
 Node *pv = new Node;
 pv->d = d;
 pv->p = 0;
 return pv;
}

// Занесення в стек
void push(Node **top, int d)
{
 Node *pv = new Node;
 pv->d = d;
 pv->p = *top;
 *top = pv;
}

// Вибірка із стеку
int pop (Node **top)
{
 int temp = (*top)->d;
 Node *pv = *top;
 *top = (*top)->p;
```

```
 delete pv;
 return temp;
}
```

Результат роботи програми:

```
5 4 3 2 1
```

14.2. Черга

Черга реалізує принцип обслуговування FIFO (first in – first out, першим прийшов, – першим пішов). Приклад черги – черга людей в магазині. Нижче приведена програма, яка формує чергу з п'яти цілих чисел і виводить її на екран. Функція поміщення елементу в кінець черги називається add, а вибірки – del. Вказівка на початок черги називається pbeg, вказівка на кінець – pend.

```
#include <iostream>
using namespace std;

struct Node
{
int d;
Node *p;
};

Node * first(int d);
void add(Node **pend, int d);
int del(Node **pbeg);

void main()
{
 Node *pbeg = first(1);
 Node *pend = pbeg;
 for (int i = 2; i<6; i++) add(&pend, i);
 while (pbeg)
   cout << del(&pbeg) << " ";
}

// Початкове формування черги
Node* first(int d)
{
 Node *pv = new Node;
 pv->d = d;
 pv->p = 0;
 return pv;
}
```

```
// Додавання в кінець
void add(Node **pend, int d)
{
 Node *pv = new Node;
 pv->d = d;
 pv->p = 0;
 (*pend)->p = pv;
 *pend = pv;
}

// Вибірка
int del(Node **pbeg)
{
 int temp = (*pbeg)->d;
 Node *pv = *pbeg;
 *pbeg = (*pbeg)->p;
 delete pv;
 return temp;
}
```

Результат роботи програми:

```
12 3 4 5
```

14.3. Лінійний список

Найпростіший спосіб зв'язати безліч елементів – зробити так, щоб кожен елемент містив посилання на наступний. Такий список називається однонаправленим (однозв’язним). Якщо додати в кожен елемент друге посилання – на попередній елемент, вийде двонаправлений список (двозв’язний), якщо останній елемент зв'язати вказівкою з першим, вийде кільцевий список.

Кожен елемент списку містить ключ, що ідентифікує цей елемент. Ключ зазвичай буває або цілим числом, або рядком і є частиною поля даних. У якості ключа в процесі роботи із списком можуть виступати різні частини поля даних. Наприклад, якщо створюється лінійний список із записів, які містять прізвище, рік народження, стаж роботи та стать, будь-яка частина запису може виступати як ключ: при впорядковувань списку за алфавітом ключем буде прізвище, а при пошуку, наприклад, ветеранів праці ключем буде стаж. Ключі різних елементів списку можуть збігатися.

Над списками можна виконувати наступні операції:

– початкове формування списку (створення першого елементу);
– додавання елементу в кінець списку;
– читання елементу із заданим ключем;
– вставка елементу в задане місце списку (до або після елементу із заданим ключем);

– видалення елементу із заданим ключем;
– впорядковування списку по ключу.

Розглянемо двонаправлений лінійний список. Для формування списку і роботи з ним потрібно мати принаймні одну вказівку – на початок списку. Зручно завести ще одну вказівку – на кінець списку. Для простоти допустимо, що список складається з цілих чисел, тобто опис елементу списку виглядає таким чином:

```
struct Node
{
int d;
Node *next;
Node *prev;
};
```

Нижче приведена програма, яка формує список з 5 чисел, додає число в список, видаляє число із списку і виводить список на екран. Вказівка на початок списку позначена pbeg, на кінець списку – pend, допоміжні вказівки – pv та pkey.

```
#include <iostream>
using namespace std;

struct Node
{
 int d;
 Node *next;
 Node *prev;
};

Node* first(int d);
void add(Node **pend, int d);
Node* find(Node * const pbeg, int i);
bool remove(Node **pbeg, Node **pend, int key);
Node* insert(Node * const pbeg, Node **pend,int key,int d);

void main()
{

// Формування першого елементу списку
Node *pbeg = first(1);
Node *pend = pbeg;
//Додавання в кінець списку чотирьох елементів 2, 3, 4 та 5
for (int i = 2; i<6; i++) add(&pend, i);
// Вставка елементу 200 після елементу 2
insert(pbeg, &pend, 2, 200);
// Видалення елементу 5
if(!remove (&pbeg, &pend, 5)) cout << "не знайдений";
Node *pv = pbeg;
```

```
while (pv)
{
// виведення списку на екран
cout << pv->d << " ";
pv = pv->next;
}
}

// Формування першого елементу
Node* first(int d)
{
 Node *pv = new Node;
 pv->d = d;
 pv->next = 0;
 pv->prev = 0;
 return pv;
}

// Додавання в кінець списку
void add(Node **pend, int d)
{
 Node *pv = new Node;
 pv->d = d;
 pv->next = 0;
 pv->prev = *pend;
 (*pend)->next = pv;
 *pend = pv;
}

// Пошук елементу по ключу
Node* find(Node * const pbeg, int d)
{
 Node *pv = pbeg;
 while (pv)
 {
  if(pv->d == d) break;
  pv = pv->next;
 }
 return pv;
}

// Видалення елементу
bool remove(Node **pbeg, Node **pend, int key)
{
 if(Node *pkey = find(*pbeg, key)) //1
 {
  if (pkey == *pbeg)
  { // 2
   *pbeg = (*pbeg)->next;
```

```
    (*pbeg)->prev =0;
   }
    else if (pkey == *pend)
    {// 3
     *pend = (*pend)->prev;
     (*pend)->next =0;
    }
   else
    { // 4
     (pkey->prev)->next = pkey->next;
     (pkey->next)->prev = pkey->prev;
    }
delete pkey;
return true; // 5
}
return false; // 6
}

// Вставка елементу
Node *insert (Node* const pbeg,Node **pend,int key,int d)
{
 if(Node *pkey = find(pbeg,key))
 {
  Node *pv = new Node;
  pv->d = d;
  // 1 - встановлення зв'язку нового вузла з наступним
  pv->next = pkey->next;
  // 2 - встановлення зв'язку нового вузла з попереднім
  pv->prev = pkey;
  // 3 - встановлення зв'язку попереднього вузла з новим
  pkey->next = pv;
  // 4 - встановлення зв'язку наступного вузла з новим
  if(pkey!=*pend) (pv->next)->prev = pv;
  // Оновлення вказівки на кінець списку
  // якщо вузол вставляється в кінець
 else *pend = pv;
  return pv;
 }
 return 0;
}
```

Результат роботи програми:

```
1 2 200 3 4
```

Всі параметри, не змінні всередині функцій, повинні передаватися з модифікатором const. Вказівки, які можуть змінитися (наприклад, при видаленні із списку останнього елементу вказівку на кінець списку потрібно скорегувати), передаються за адресою.

Розглянемо докладніше функцію видалення елементу із списку remove. Її параметрами є вказівки на початок і кінець списку і ключ елементу, який треба видалити. У рядку 1 виділяється пам'ять під локальну вказівку pkey, якій привласнюється результат виконання функції знаходження елементу по ключу find. Ця функція повертає вказівку на елемент у разі успішного пошуку та 0, якщо елементу з таким ключем в списку немає. Якщо pkey набуває ненульового значення, умова в операторі if стає істинною (елемент існує), і управління передається оператору 2, якщо немає – виконується повернення з функції із значенням false (оператор 6).

Видалення із списку відбувається по-різному залежно від того, знаходиться елемент на початку списку, в середині або в кінці. У операторі 2 перевіряється, чи знаходиться елемент, що видаляється, на початку списку – в цьому випадку слід скорегувати вказівку pbeg на початок списку так, щоб вона вказувала на наступний елемент в списку, адреса якого знаходиться в полі next першого елементу. Новий початковий елемент списку повинен мати в своєму полі вказівку на попередній елемент значення 0.

Якщо елемент, що видаляється, знаходиться в кінці списку (оператор 3), потрібно змістити вказівку pend кінця списку на попередній елемент, адресу якого можна отримати з поля prev останнього елементу. Крім того, потрібно обнулити для нового останнього елементу вказівку на наступний елемент. Якщо видалення походить з середини списку, то єдине, що треба зробити, – забезпечити двобічний зв'язок попереднього і наступного елементів. Після корегування вказівок пам'ять з-під елементу звільняється, і функція повертає значення true.

Контрольні питання

1. Що таке динамічні структури даних?
2. Які динамічні структури вам відомі?
3. Розкрийте сутність роботи із стеком.
4. Наведіть приклад початкового формування стеку.
5. Наведіть приклад занесення даних в стек.
6. Наведіть приклад вибірки даних із стеку.
7. Розкрийте сутність роботи з чергою.
8. Наведіть приклад початкового формування черги.
9. Наведіть приклад додавання даних в кінець черги.
10. Наведіть приклад вибірки даних з черги.
11. Розкрийте сутність роботи з лінійним списком.
12. Наведіть приклад формування першого елементу списку.
13. Наведіть приклад додавання даних в кінець списку.
14. Наведіть приклад пошуку елементів в списку по ключу.
15. Наведіть приклад вставки елементу в список.
16. Наведіть приклад видалення елементу зі списку.

Завдання

У даній роботі необхідно зробити програму з використанням динамічних структур даних, а саме стеків, черг, лінійних списків.

Варіанти індивідуальних завдань

1. Є стек цілих чисел, який складається з n елементів. Визначити добуток додатних парних чисел стеку Якщо таких елементів немає, вивести повідомлення: «Додатних парних елементів в стеку немає».
2. Є черга цілих чисел, яка складається з n елементів. Визначити добуток від'ємних чисел черги, що стоять на парних позиціях. Якщо таких елементів немає, вивести повідомлення: «Від'ємних елементів в черзі немає».
3. Є список цілих чисел, який складається з n елементів. Визначити суму і кількість непарних чисел списку. Якщо таких елементів немає, вивести повідомлення: «Непарних елементів в списку немає».
4. Є стек цілих чисел, який складається з n елементів. Визначити в ньому суму і кількість чисел, кратних 5. Якщо таких елементів немає, вивести повідомлення: «Елементів кратних 5 в стеку немає».
5. Є черга цілих чисел, яка складається з n елементів. Визначити у ній кількість чисел, кратних 2, і чисел, не кратних 3. Якщо таких елементів немає, вивести повідомлення: «Чисел, кратних 2 і чисел не кратних 3 в черзі немає».
6. Є список цілих чисел, який складається з n елементів. Визначити кількість і добуток елементів списку, які знаходяться в діапазоні [0; 7]. Якщо таких елементів немає, вивести повідомлення: «Елементів з діапазону [0; 7] в списку немає».
7. Є стек цілих чисел, який складається з n елементів. У ньому визначити середнє арифметичне значення чисел, кратних трьом. Якщо таких елементів немає, вивести повідомлення: «Елементів кратних 3 в стеку немає».
8. Є черга цілих чисел, яка складається з n елементів. Визначити середнє арифметичне значення елементів черги, які містяться в діапазоні [-3; 5]. Якщо таких елементів немає, вивести повідомлення: «Елементів, що задовольняють вимозі, в черзі немає».
9. Є список цілих чисел, який складається з n елементів. У ньому визначити мінімальний елемент та його порядковий номер.
10. Є стек цілих чисел, який складається з n елементів. Визначити добуток непарних елементів стеку, що стоять на парних позиціях. Якщо таких елементів немає, вивести повідомлення: «Непарних елементів, що стоять на парних позиціях в стеку немає».
11. Є черга цілих чисел, яка складається з n елементів. Визначити середнє арифметичне значення елементів черги, кратних восьми. Якщо таких елементів немає, вивести повідомлення: «Елементів кратних 8 в черзі немає».

12. Є список цілих чисел, який складається з n елементів. У ньому серед додатних елементів визначити максимальний елемент.

13. Є стек цілих чисел, який складається з n елементів. Визначити добуток додатних елементів стеку та їх кількість. За відсутності додатних чисел вивести повідомлення «Додатних чисел в стеку немає».

14. Є черга цілих чисел, яка складається з n елементів. Обчислити кількість елементів черги, кратних семи. За відсутності таких елементів вивести повідомлення «Елементів кратних 7 немає».

15. Є список цілих чисел, який складається з n елементів. Визначити середнє арифметичне значення мінімального і максимального елементів списку.

16. Є стек цілих чисел, який складається з n елементів. Визначити суму і добуток додатних чисел, що стоять на парних позиціях.

17. Є черга цілих чисел, яка складається з n елементів. Визначити середнє арифметичне значення додатних елементів і середнє арифметичне значення від'ємних елементів.

18. Є список цілих чисел, який складається з n елементів. Визначити окремо кількість від'ємних чисел, кількість додатних чисел і чисел, рівних нулю.

19. Є стек цілих чисел, який складається з n елементів. Визначити суму додатних парних чисел стеку. У разі відсутності додатних парних чисел вивести повідомлення «Додатних парних чисел в стеку немає».

20. Є черга цілих чисел, яка складається з n елементів. Визначити середнє арифметичне значення чисел черги, що стоять на парних позиціях, і середнє арифметичне значення чисел, що стоять на непарних позиціях.

21. Є список цілих чисел, який складається з n елементів. Визначити суму додатних елементів списку, що стоять на позиціях, кратних трьом.

22. Є стек цілих чисел, який складається з n елементів. Обчислити добуток максимального і мінімального елементів стеку.

23. Є черга цілих чисел, яка складається з n елементів. Визначити різницю між максимальним та мінімальним елементами.

24. Є список цілих чисел, який складається з n елементів. Обчислити добуток та кількість від'ємних непарних чисел, розташованих на парних позиціях.

25. Є стек цілих чисел, який складається з n елементів. Обчислити суму та кількість чисел, що діляться без остачі на 5. Якщо таких чисел немає, вивести повідомлення «Чисел, що діляться без остачі на 5, в стеку немає».

РОЗДІЛ 15. ТИПОВІ МЕТОДИ СОРТУВАННЯ МАСИВІВ

До основних типових алгоритмів сортування належать: бульбашкове сортування, сортування за допомогою вибору, сортування вставками.

До основних покращених алгоритмів сортування належать: сортування Шелла, та швидке сортування (quicksort). Найшвидшим алгоритмом є алгоритм швидкого сортування. Його доцільно використовувати для великих об'ємів даних. Якщо дані займають невеликі обсяги можна використовувати типові алгоритми сортування. Розглянемо кожен з алгоритмів більш детально.

15.1. Бульбашкове сортування (bubble sort)

Бульбашкове сортування (bubble sort, сортування методом бульбашки, або просто сортування бульбашкою). Метод бульбашкового сортування є самим простим та малоефективним.

Бульбашкове сортування відноситься до класу обмінних сортувань, тобто до класу сортувань методом обміну. Її алгоритм містить порівняння (тобто багатократні порівняння одних і тих же елементів), що повторюються, і, при необхідності, обмін сусідніх елементів. Елементи поводяться подібно до бульбашок повітря у воді - кожний з них піднімається на свій рівень. Проста форма алгоритму наступна:

```
void BulbSort(double* a, int l, int r)
{
int i,j;
double tmp;

for(i=l;i<r;i++)
    for(j=r;j>i;j--)
     if(a[j]<a[i])
          {
           tmp=a[i];
           a[i]=a[j];
           a[j]=tmp;
          }
}
```

Тут **a** – вказівка на масив типу **double**, що підлягає сортуванню, **l** – нижній індекс, **r** – верхній індекс масиву. Робота бульбашкового сортування виконується в двох циклах. Зовнішній цикл переглядається до передостаннього елементу масиву (**i<r**). Це забезпечує розміщення елементів в правильному порядку до кінця виконання функції навіть в найгіршому випадку. Всі порівняння і обміни виконуються у внутрішньому циклі. (Злегка покращувана версія алгоритму бульбашкового сортування завершує роботу, якщо при прогляданні масиву не було зроблено жодного обміну, але це досягається за рахунок додавання ще одного порівняння при кожному проході внутрішнього

циклу.)

При аналізі будь-якого алгоритму сортування корисно знати, скільки операцій порівняння і обміну буде виконано в кращому, середньому і гіршому випадках. Оскільки характеристики виконуваного коду залежать від таких чинників, як оптимізація, вироблювана компілятором, відмінності між процесорами і особливості реалізації, ми не намагатимемося набути точного значення цих параметрів. Натомість сконцентруємо свою увагу на загальній ефективності кожного алгоритму.

У бульбашковому сортуванні кількість порівнянь завжди одна й та ж, оскільки два цикли for повторюються вказану кількість разів незалежно від того, був список спочатку впорядкований чи ні. Це означає, що алгоритм бульбашкового сортування завжди виконує $(n^2 - n)/2$ порівнянь, де n - кількість сортованих елементів. Дана формула виведена на тій підставі, що зовнішній цикл виконується n - 1 раз, а внутрішній виконується в середньому $n/2$ раз. Добуток цих величин і дає попередній вираз. Зверніть увагу на член n^2 у формулі. Бульбашкове сортування є алгоритмом *порядку* n^2, оскільки час його виконання пропорційний квадрату кількості сортованих елементів. Необхідно визнати, що алгоритм порядку n^2 не ефективний при великій кількості елементів, оскільки час виконання росте експоненціально залежно від кількості сортованих елементів.

У алгоритмі бульбашкового сортування кількість обмінів в кращому разі рівна нулю, якщо масив вже відсортований. Проте в середньому і гіршому випадках кількість обмінів також є величиною порядку n^2.

15.2. Сортування за допомогою вибору (choice sort)

При сортуванні за допомогою вибору (сортування вибором, сортування вибірками) з масиву вибирається елемент з найменшим значенням і обмінюється з першим елементом. Потім з тих, що залишилися n - 1 елементів знову вибирається елемент з найменшим ключем і обмінюється з другим елементом, і т.д. Ці обміни продовжуються до двох останніх елементів. Наприклад, якщо застосувати метод вибору до масиву 4, 3, 1, 2, кожен прохід буде виглядати так, як показано нижче:

Початок 4 3 1 2
Прохід 1 1 3 4 2
Прохід 2 1 2 4 3
Прохід 3 1 2 3 4

Нижченаведений код демонструє просте сортування за допомогою вибору:

```
void ChoiceSort(double* a, int l, int r)
{
   int i,j,k,exch;
   double tmp;

   for(i=l;i<=r;i++)
   {
    k=i;
    exch=0;
    tmp=a[i];
      for(j=i+1;j<=r;j++)
      {
       if(a[j]<tmp)
          { k=j;tmp=a[j];exch=1;}
      }
     if(exch){a[k]=a[i];a[i]=tmp;}
   }
}
```

Як і в бульбашковому сортуванні, зовнішній цикл виконується *n* - 1 раз, внутрішній - в середньому *n/2* раз. Отже, сортування за допомогою вибору вимагає $(n^2 - n)/2$ порівняння. Таким чином, це алгоритм порядку n^2, із-за чого він вважається дуже повільним для сортування великої кількості елементів. Не дивлячись на те, що кількість порівнянь в бульбашковому сортуванні і сортуванні за допомогою вибору однакове, в останній кількість обмінів в середньому випадку набагато менше, ніж в бульбашковому сортуванні.

15.3. Сортування вставками (insert sort)

Сортування вставками – третій і останній з простих алгоритмів сортування. Спочатку він сортує два перші елементи масиву. Потім вставляється третій елемент у відповідну порядку позицію по відношенню до перших двох елементів. Після цього вставляється четвертий елемент в список з трьох елементів. Цей процес повторюється до тих пір, поки не будуть вставлені всі елементи. Наприклад, при сортуванні масив 4, 3, 1, 2 кожен прохід алгоритму виглядатиме таким чином:

Початок	4 3 1 2
Прохід 1	3 4 1 2
Прохід 2	1 3 4 2
Прохід 3	1 2 3 4

Приклад реалізації сортування вставками показаний нижче:

```
void InsertSort(double* a, int l, int r)
{
  int i,j;
  double tmp;

  for(i=l+1;i<=r;i++)
  {
   tmp=a[i];
     for(j=i-1;(j>=l) && (tmp<a[j]);j--)
              a[j+1]=a[j];
   a[j+1]=tmp;
  }
}
```

На відміну від бульбашкового сортування і сортування за допомогою вибору, кількість порівнянь в сортуванні вставками залежить від початкової впорядкованості списку. Якщо список вже відсортований, кількість порівнянь дорівнює n - 1; інакше його продуктивність є величиною порядку n^2.

Взагалі кажучи, в гірших випадках сортування вставками настільки ж неефективне, як бульбашкове сортування і сортування за допомогою вибору, а в середньому вона лише трохи краще. Проте, у сортуванні вставками є переваги. Його поведінка є природною. Іншими словами, воно працює менше всього, коли масив вже впорядкований, і більше всього, коли масив відсортований в зворотному порядку. Тому сортування вставками – ідеальний алгоритм для майже впорядкованих списків.

Не дивлячись на те, що кількість порівнянь при певних наборах даних може бути досить низькою, при кожній вставці елементу на своє місце масив необхідно зрушувати. Внаслідок цього кількість переміщень може бути значною.

Покращені алгоритми сортування.

Всі прості алгоритми мають один фатальний недолік – час їх виконання має порядок n^2. Це робить сортування великих об’ємів даних дуже повільним. По суті, в якийсь момент ці алгоритми стають дуже повільними, щоб їх застосовувати. Тобто необхідно застосовувати кращі алгоритми сортування.

Найбільш популярні два покращені методи сортування. Перший називається *сортуванням Шелла.* Другий - *швидке сортування* - вважається найкращим алгоритмом сортування. Обидва методи є більш довершеними способами сортування і мають набагато кращу загальну продуктивність, чим будь-який з простих методів.

15.4. Сортування Шелла

Сортування Шелла називається так на ім’я свого автора, Дональда Л. Шелла (відкрите в 1959 р.). Проте ця назва закріпилася тому, що дія цього методу часто ілюструється рядами морських раковин, які перекривають одна одну (по-англійськи «shell» - «раковина»). Загальна ідея запозичена з сортування вставками і ґрунтується на зменшенні кроків (крок – це відстань між сортованими елементами на конкретному етапі сортування). Розглянемо діаграму (рис. 15.1). Спочатку сортуються всі елементи, віддалені один від одного на три позиції. Потім сортуються елементи, розташовані на відстані двох позицій. Нарешті, сортуються всі сусідні елементи.

Те, що цей метод дає гарні результати, або навіть те, що він взагалі сортує масив, побачити не так просто. Проте, це вірно. Кожен прохід сортування розповсюджується на відносно невелику кількість елементів або на елементи, розташовані вже у відносному порядку. Тому сортування Шелла ефективне, а кожен прохід підвищує впорядкованість, тобто зменшує кількість безладів (інверсій).

Конкретна послідовність кроків може бути і інша. Єдине правило полягає в тому, щоб останній крок був рівний 1. Наприклад, така послідовність: 9, 5, 3, 2, 1 дає ефективні результати і застосовується у показаній тут реалізації сортування Шелла. Слід уникати послідовностей, які є ступенями числа 2 – по математично складних міркуваннях вони зменшують ефективність сортування (але сортування буде працювати).

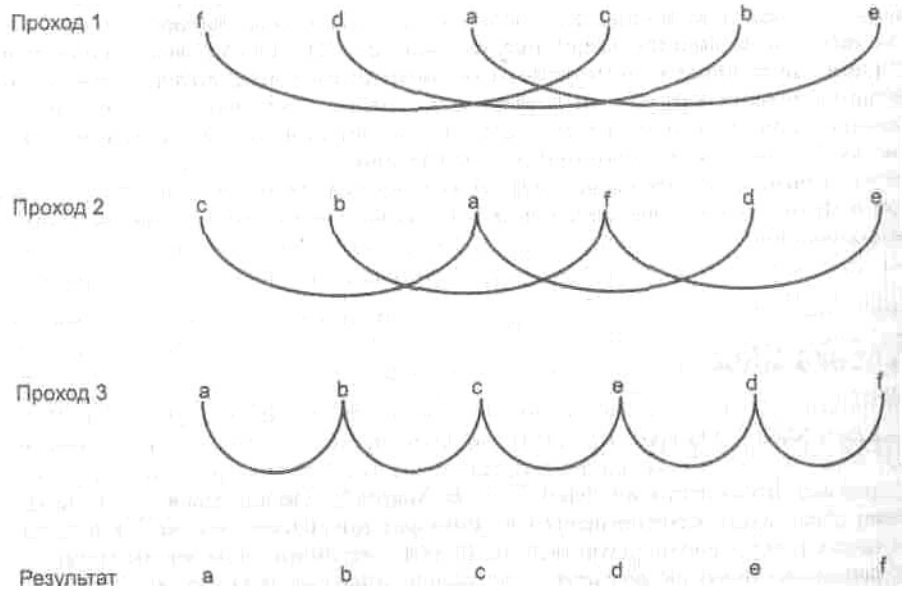


Рис.15.1 Сортування Шелла

```
void ShellSort(double* items, int l, int r)
{
  int i,j,gap,k;
  double x,a[5];
  a[0]=9;a[1]=5;a[2]=3;a[3]=2; a[4]=1;
  for(k=0;k<5;k++)
 {
  gap=a[k];
    for(i=gap;i<=r;i++)
   {
    x=items[i];
       for(j=i-gap;(x<items[j]) && (j>=l);j-=gap)
           items[j+gap]=items[j];
    items[j+gap]=x;
   }
 }
}
```

Аналіз сортування Шелла пов'язаний з дуже складними математичними задачами. Час сортування пропорційний $n^{1,2}$ при сортуванні n елементів. А це вже істотне поліпшення в порівнянні з сортуваннями порядку n^2. Проте, алгоритм швидкого сортування ще ефективніший.

15.5. Швидке сортування (quick sort)

Швидке сортування, придумане Ч. А. Р. Хоаром (Charles Antony Richard Hoare) і названо його ім'ям, вважається кращим з існуючих в даний час алгоритмів сортування загального призначення. У його основі лежить сортування обмінами.

Швидке сортування побудоване на ідеї ділення. Загальна процедура полягає в тому, щоб вибрати деяке значення, назване *компарандом* (*comparand*, операнд в операції порівняння, іноді називається також основою або критерієм розбиття), а потім розбити масив на дві частини. Всі елементи, більші або рівні *компаранду*, переміщаються на одну сторону, а менші – на іншу. Потім цей процес повторюється для кожної частини до тих пір, поки масив не буде відсортований. Наприклад, якщо початковий масив складається з чисел 6, 5, 4, 1, 3, 2, а у якості *компаранди* використовується число 4, перший прохід швидкого сортування переупорядкує масив таким чином:

Початок 6 5 4 1 3 2
Прохід 1 2 3 1 4 5 6

Потім сортування повторюється для обох половин масиву, тобто 2 3 1 і 4 5 6. Як видно, цей процес за своєю суттю рекурсивний, і, дійсно, в чистому вигляді швидке сортування реалізується як рекурсивна функція.

Значення *компаранди* можна вибирати двома способами – випадковим чином або усереднивши невелику кількість значень з масиву. Для

оптимального сортування необхідно вибирати значення, яке розташоване точно у середині діапазону всіх значень. Проте для більшості наборів даних це зробити непросто. У гіршому разі вибране значення виявляється одним з крайніх. Проте, навіть в цьому випадку швидке сортування працює правильно. У приведеній нижче версії швидкого сортування у якості *компаранди* вибирається середній елемент масиву.

```
void QuickSort(double * items, int l, int r)
{
  int i,j;
  double x,y;
  i=l;j=r;
  x=items[(l+r)/2];
  do
 {
  while((items[i]<x) && (i<r)) i++;
  while((x<items[j]) && (j>l)) j--;
  if(i<=j)
  {
   y=items[i];
   items[i]=items[j];
   items[j]=y;
   i++;j--;
  }
 }while(i<=j);
 if(l<j) QuickSort(items,l,j);
 if(i<r) QuickSort(items,i,r);
}
```

Середня кількість порівнянь дорівнює $n \log n$, а середня кількість обмінів приблизно рівна $n/6 \log n$. Ці величини набагато менше відповідних характеристик розглянутих раніше алгоритмів сортування.

Необхідно згадати про один особливо проблематичний аспект швидкого сортування. Якщо значення *компаранди* в кожному діленні дорівнює найбільшому значенню, швидке сортування стає «повільним сортуванням» з часом виконання порядку n^2. Тому треба уважно вибрати метод визначення *компаранди*. Цей метод часто визначається природою сортованих даних. Наприклад, в дуже великих списках поштової розсилки, в яких сортування відбувається по поштовому індексу, вибір простий, тому що поштові індекси досить рівномірно розподілені – *компаранду* можна визначити за допомогою простої функції алгебри. Проте в інших базах даних часто кращим вибором є випадкове значення.

Кожен програміст повинен мати в своєму розпорядженні широкий набір алгоритмів сортування. Не дивлячись на те, що в середньому випадку оптимальним є саме швидке сортування, воно не є кращим у всіх випадках. Наприклад, при сортуванні дуже маленьких списків (наприклад, менше 100

елементів) додатковий об’єм роботи створюваний рекурсивними викликами швидкого сортування, може перекрити переваги її більш ефективного алгоритму. У таких окремих випадках один з простих методів сортування – можливо, навіть бульбашкове сортування – може працювати швидше. Крім того, якщо відомо, що список вже майже впорядкований або якщо немає потреби переставляти однакові елементи, який-небудь інший алгоритм підійде краще ніж швидке сортування. Швидке сортування є кращим алгоритмом загального призначення, але це не означає що в конкретних випадках інші підходи не дадуть кращих результатів.

Наведемо порівняльну таблицю алгоритмів сортування (табл. 15.1). Порівняння методів сортування виконувалась на окремому персональному комп’ютері, тому час виконання кожного з алгоритмів вказаних в таблиці слід розглядати як відносну величину, тобто тільки для порівняння швидкодії та ефективності алгоритмів між собою.

Таблиця 15.1

Час порівняння методів сортування в залежності від кількості елементів масиву

Метод сортування	**Час (секунди) для відповідної кількості елементів**		
	50000	**100000**	**200000**
бульбашкове	*9,078*	*37,25*	*152,094*
вибором	*3,796*	*15,625*	*64,25*
вставками	*1,969*	*8,562*	*45,079*
Шелла	*0,437*	*2,922*	*33,375*
швидке	*0,007*	*0,015*	*0,031*

Найефективнішим алгоритмом сортування серед усіх методів є алгоритм швидкого сортування (quick sort); серед типових алгоритмів сортування – алгоритм вставками (insert sort).

Контрольні питання

1. Які методи сортування масивів ви знаєте?
2. Розкрийте сутність бульбашкового методу сортування. Наведіть приклад.
3. Розкрийте сутність методу сортування за дпомогою вибору. Наведіть приклад.
4. Розкрийте сутність методу сортування вставками. Наведіть приклад.
5. Розкрийте сутність методу сортування Шелла. Наведіть приклад.
6. Розкрийте сутність методу швидкого сортування. Наведіть приклад.

Завдання

Скласти програму з використанням типових методів сортування.

Варіанти індивідуальних завдань

1. В заданій матриці A(m, n) упорядкувати елементи кожного рядка по зростанню. Використати метод сортування за допомогою вибору. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

2. Розмістити елементи кожного рядка матриці A(m, n) у порядку зменшення значень. Використати метод сортування вставками. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

3. В заданій матриці A(m, n) упорядкувати елементи кожного стовпця по зростанню. Використати метод швидкого сортування. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

4. Розмістити елементи кожного стовпця матриці A(m, n) у порядку зменшення значень. Використати метод бульбашкового сортування. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

5. Розмістити елементи головної діагоналі квадратної матриці A(n, n) у порядку зменшення значень. Використати метод швидкого сортування. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

6. Розмістити елементи головної діагоналі квадратної матриці A(n, n) по зростанню. Використати метод сортування за допомогою вибору. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

7. Розмістити елементи на неголовній діагоналі квадратної матриці A(n, n) в порядку їх зменшення. Використати метод сортування Шелла. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

8. Розмістити елементи на неголовній діагоналі квадратної матриці A(n, n) в порядку їх зростання. Використати метод сортування вставками. Вхідні

дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

9. Задано матрицю A(m, n). Необхідно упорядкувати рядки матриці по зростанню сум елементів рядків. Використати метод бульбашкового сортування. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

10. Задано матрицю A(m, n). Необхідно упорядкувати рядки матриці в порядку зменшення сум елементів рядків. Використати метод сортування за допомогою вибору. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

11. Задано матрицю A(m, n). Необхідно упорядкувати рядки матриці по зростанню сум елементів стовпців. Використати метод сортування Шелла. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

12. Задано матрицю A(m, n). Необхідно упорядкувати рядки матриці в порядку зменшення сум елементів стовпців. Використати метод швидкого сортування. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

13. Задано матрицю A(m, n). Необхідно упорядкувати рядки матриці по зростанню добутків елементів рядків. Використати метод сортування вставками. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

14. Задано матрицю A(m, n). Необхідно упорядкувати рядки матриці в порядку зменшення добутків елементів рядків. Використати метод сортування за допомогою вибору. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку FILE, та результат записати до файлу rezult.txt.

15. Задано матрицю A(m, n). Необхідно упорядкувати рядки матриці по зростанню добутків елементів стовпців. Використати метод сортування Шелла. Вхідні дані зчитати з послідовного файлу vvod.txt за допомогою потоку fstream, та результат записати до файлу rezult.txt.

РОЗДІЛ 16. ЧИСЕЛЬНЕ ДИФЕРЕНЦІЮВАННЯ ТА ІНТЕГРУВАННЯ

16.1. Методи правих та центральних різниць чисельного диференціювання

Приріст аргументу і приріст функції. Нехай функція $y = f(x)$ визначена в точках x_0 та $x_1 = x_0 + \Delta x$. Різниця $x_1 - x_0 = \Delta x$ називається приростом аргументу, а різниця $f(x_1) - f(x_0) = f(x_0 + \Delta x) - f(x_0)$ називається приростом функції при переході від значення аргументу x_0 до аргументу $x_1 = x_0 + \Delta x$. Приріст функції позначається Δf або Δy, тобто $\Delta f = f(x_0 + \Delta x) - f(x_0)$.

Похідною функції $f(x)$ в точці x_0 (позначають $f'(x_0)$) називається межа відношення приросту функції до приросту аргументу при умові, що приріст аргументу прямує до нуля, тобто

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x},$$

де $\Delta x = x_1 - x_0$ – приріст аргументу; x_1 і x_0 – два значення незалежної змінної з області визначення функції $f(x)$; $f(x_0 + \Delta x) - f(x_0) = \Delta f$ – приріст функції в точці x_0.

Наприклад, якщо $f(x) = 3x^2 + 2$, то $f'(x_0) =$

$$= \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} = \lim_{\Delta x \to 0} \frac{3(x_0 + \Delta x)^2 + 2 - 3x_0^2 - 2}{\Delta x} = \lim_{\Delta x \to 0} \frac{3(x_0^2 + 2x_0 \Delta x + (\Delta x)^2) - 3x_0}{\Delta x} =$$

$$= \lim_{\Delta x \to 0} \frac{6x_0 \Delta x + 3(\Delta x)^2}{\Delta x} = \lim_{\Delta x \to 0} (6x_0 + 3\Delta x) = 6x_0 + 0 = 6x_0 .$$

Функція, яка має похідну в точці x_0, називається такою, що диференціюється в цій точці. Якщо функція має похідну в кожній точці деякого проміжку, то вона диференційована на цьому проміжку. Похідна функції $f(x)$, що диференціюється на проміжку, сама є функцією аргументу x.

Похідна широко використовується в різних методах аналізу даних, причому від ефективності обчислення похідної залежить ефективність, а часто і сама можливість реалізації вживаних методів аналізу. Похідні можуть визначатися аналітично або чисельно.

Кінцево-різницеві схеми чисельного визначення похідних побудовані на ідеї застосування такого малого приросту аргументу (похитування), що обчислення дає достатню для практичного застосування точність. Формула методу правих різниць є класичною формулою визначення похідної і виглядає наступним чином:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h},$$

а метод центральних різниць може бути записаний як

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h},$$

де в обох випадках h – кінцево-різницевий інтервал (крок похитування).

Крок похитування h повинен наближуватись до нуля (дорівнює маленькій величині).

Наведемо фрагмент програми чисельного диференціювання.

```
#include <iostream>
using namespace std;

double func(double x)
{
 return x*x + 5*x + 6;
}

void main()
{
 double x,h,t1,t2;
 h = 0.0000000001;
 x = 3;

 t1 = (func(x+h)-func(x))/h;
 t2 = (func(x+h)-func(x-h))/(2*h);

 cout<<"\nПохідна (метод правих різниць) "<<t1;
 cout<<"\nПохідна (метод центральних різниць) "<<t2;
}
```

У даному фрагменті на прикладі функції $y = x^2 + 5x + 6$ реалізовано методи правих та центральних різниць, відповідно змінні **t1** та **t2**. Результат виконання програми при значенні аргументу $x = 3$ виглядатиме наступним чином:

```
Похідна (метод правих різниць) 11
Похідна (метод центральних різниць) 11
```

16.2. Методи прямокутників, трапецій, Сімпсона (парабол) чисельного інтегрування

Задача обчислення визначеного інтеграла, тобто вирази вигляду

$$\int_a^b f(x)dx,$$

часто виникає в аналізі даних. Числа a і b називають відповідно нижньою і верхньою межами інтеграції, а відрізок $[a, b]$ називається ділянкою інтеграції. Функція $f(x)$ називається підінтегральною функцією. Якщо підінтегральна функція безперервна на ділянці інтеграції і одна межа або обидві відразу покладені рівними ∞, а також якщо підінтегральна функція має один або декілька розривів на ділянці інтеграції, інтеграл називається невласним. Невласний інтеграл сходиться (існує), якщо існує межа інтеграла в даних випадках, інакше говорять, що невласний інтеграл розходиться (не існує). Невласні інтеграли часто виникають, наприклад, при обчисленні теоретичних функцій розподілу. Із зменшенням або збільшенням, залежно від типу розподілу, меж інтеграції ці функції зменшуються дуже швидко, тому

практично обчислення можна здійснювати по тих же алгоритмах, що і для звичайних визначених інтегралів, тільки відповідна нескінченній нижній або верхній межі величина підбирається, відповідно, достатньо малою або достатньо великою для того, щоб точність обчислень була допустимою. Типовими для обчислення визначених інтегралів є методи прямокутників, трапецій, Сімпсона (парабол).

Формула за методом прямокутників має наступний вигляд:

$$\int_a^b f(x)dx \approx h\sum_{i=0}^{n-1} f(x_i) \approx h\sum_{i=1}^{n} f(x_i),$$

де $h = (b - a)/n$ – крок інтеграції;
n – задане число ділянок розбиття області інтегрування;
$x_i = a + ih$, $i = 0, 1, 2 ., n$.

Формула за методом трапецій виглядає наступним чином:

$$\int_a^b f(x)dx \approx h\left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right].$$

Формула за методом Сімпсона (парабол) представлена в наступному вигляді:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a)+f(b)+2\left(f(x_2)+f(x_4)+...+f(x_{n-2})\right)+4\left(f(x_1)+f(x_3)+...+f(x_{n-1})\right)\right],$$

де n – парне.

Наведемо фрагмент програми реалізації методів чисельного інтегрування.

```
#include <iostream>
using namespace std;

double func(double x)
{
 return x*x + 5*x + 6;
}
void main()
{
 double a,b,h,x;
 double s,s1,s2,t1,t2,t3;

 int i,n;

 a = 1;
 b = 4;
 n = 100;
 h = (b-a)/n;


 // Метод прямокутників
s = 0;
   for(i=0;i<n;i++)
```

```
    {
      x = a + i*h;
      s+= func(x);
     }

t1 = s*h;
   cout<<"\nІнтеграл (метод прямокутників) "<<t1;

 // Метод трапецій
s = 0;
   for(i=1;i<n;i++)
    {
      x = a + i*h;
      s+= func(x);
     }

t2 = ((func(a)+func(b))/2+s)*h;
   cout<<"\nІнтеграл (метод трапецій) "<<t2;

 // Метод Сімпсона (парабол)
s1 = s2 = 0;
   for(i=1;i<n;i++)
    {
      x = a + i*h;
      if (i&1)s1+= func(x);
         else s2+= func(x);
     }

t3 = (func(a)+func(b)+2*s2+4*s1)*h/3;
   cout<<"\nІнтеграл (метод Сімпсона) "<<t3;
}
```

Для інтегрування вибрано функцію $y = x^2 + 5x + 6$, межі інтегрування: $a = 1$, $b = 4$. Фігура обмежена цими величинами розбивається на 100 частин ($n = 100$). Слід зауважити, що для реалізації методу Сімпсона (парабол) число n повинне бути парним.

Результат виконання програми:

```
Інтеграл (метод прямокутників)     76,0505
Інтеграл (метод трапецій)          76,5005
Інтеграл (метод Сімпсона)          76,5
```

Найточнішим методом є метод Сімпсона (метод парабол).

Контрольні питання

1. Що таке похідна?
2. Розкрийте сутність методу правих різниць визначення похідної. Наведіть фрагмент програми.
3. Розкрийте сутність методу центральних різниць визначення похідної. Наведіть фрагмент програми.
4. Розкрийте сутність методу прямокутників обчислення визначеного інтегралу. Наведіть фрагмент програми.
5. Розкрийте сутність методу трапецій обчислення визначеного інтегралу. Наведіть фрагмент програми.
6. Розкрийте сутність методу Сімпсона (парабол) обчислення визначеного інтегралу. Наведіть фрагмент програми.

Завдання

Чисельне диференціювання

Скласти програму обчислення похідної в точці x_0 за допомогою методів правих та центральних різниць.

Варіанти індивідуальних завдань

1. У точці $x_0 = 2$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{\sqrt{1.2 + \ln x}}{2} e^x$. Порівняти отримані значення.
2. У точці $x_0 = 3.5$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = 0.1^2 + 5\sin x + 2.3 \cdot e^{\cos x}$. Порівняти отримані значення.
3. У точці $x_0 = 4$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = 0.5 - x^{\sin\frac{x}{2}} + \ln x$. Порівняти отримані значення.
4. У точці $x_0 = 1$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{3}{2x^2 + x + 0.6} \sin x$. Порівняти отримані значення.

5. У точці x_0 = -2.5 обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{3}{0.5x+3} + \sin x^2 + 15.2$. Порівняти отримані значення.

6. У точці x_0 = 5 обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \left(\left(\frac{3}{x} \right)^{\frac{1}{5}} - 1 \right) \sin x$. Порівняти отримані значення.

7. У точці x_0 = 3 обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{0.5}{\sin x(x+1.1)} ctg\, x - \ln x$. Порівняти отримані значення.

8. У точці x_0 = 1 обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{e^x}{\sin x} + \ln x$. Порівняти отримані значення.

9. У точці x_0 = 0 обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = -\frac{2}{\pi} \cdot \ln \left| \frac{\sqrt{\pi^2 + x^2}}{\sqrt{\pi^2 - x^2}} \right|$. Порівняти отримані значення.

10. У точці x_0 = -2 обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{\sin 2x + \cos x}{\lg(x^2 + 2)} e^x$. Порівняти отримані значення.

11. У точці x_0 = 2 обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{1 + \cos(x-2)}{\frac{x^4}{2} + \sin^2 x}$. Порівняти отримані значення.

12. У точці x_0 = 3 обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = (\sin^2(\pi x) - 2)^{2x} - tg(\sin^2(\pi x) - 2)^{4x}$. Порівняти отримані значення.

13. У точці x_0 = -2.2 обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \sqrt[3]{2 \cdot \ln(\pi - \sin^2 x)^2} - \sqrt[3]{\lg(\pi - \sin^2 x)^4}$. Порівняти отримані значення.

14. У точці x_0 = 4 обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = e^{2x+1} \cdot \frac{6.3 \cdot ctgx}{\sin^2 x}$. Порівняти отримані значення.

15. У точці $x_0 = -2$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \lg\left|4 - \cos^2\frac{x}{4}\right| - 2\ln\left(4 - \cos^2\frac{x}{4}\right)^2$. Порівняти отримані значення.

16. У точці $x_0 = 4$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{e^{2x} + e^{0.57x}}{\lg x^2} + \sin^2 x$. Порівняти отримані значення.

17. У точці $x_0 = 10$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \ln x - \frac{x - 100}{0.13 \cdot \sin x} + 8.2 \cdot e^x$. Порівняти отримані значення.

18. У точці $x_0 = 2$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{ctgx}{1 + \lg(2x)} + e^x$. Порівняти отримані значення.

19. У точці $x_0 = 7$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = x\left(\ln x + \frac{1}{100}\right)^2 - \sin^2 x$. Порівняти отримані значення.

20. У точці $x_0 = 3.5$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{(x+1)^{\frac{\pi}{2}}}{2x} + \ln^2 x$. Порівняти отримані значення.

21. У точці $x_0 = 2$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{\sqrt[3]{x^2 \cos^2 x}}{13.2 \cdot \lg(x^2 + 1.1)}$. Порівняти отримані значення.

22. У точці $x_0 = 4$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{\lg(x^2 + x^4 + 1)}{\sqrt{2x^2 + \cos^2 x}}$. Порівняти отримані значення.

23. У точці $x_0 = 1$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{\sin^4 x + ctg^2 x - \ln x}{e^x}$. Порівняти отримані значення.

24. У точці $x_0 = 3$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = (1 + x) \cdot \frac{x + \frac{x}{x^2 + 4}}{e^{-x-2} + \frac{1}{x^2 + 4}}$. Порівняти отримані значення.

25. У точці $x_0 = 2.5$ обчислити значення похідної функції, використовуючи методи правих та центральних різниць: $y = \frac{\sqrt{\lg(x^4)}}{2e^x \sin x}$. Порівняти отримані значення.

Чисельне інтегрування

Скласти програму обчислення визначеного інтегралу за допомогою методів прямокутників, трапецій, Сімпсона (парабол). Порівняти отримані результати.

Варіанти індивідуальних завдань

1. $\int_{1}^{2} \frac{dx}{\sqrt{2x^2+1,3}}$;
2. $\int_{0,2}^{1,2} \frac{dx}{\sqrt{x^2+1}}$
3. $\int_{0,8}^{1,4} \frac{dx}{\sqrt{2x^2+3}}$
4. $\int_{0,4}^{1,2} \frac{dx}{\sqrt{2+0,5x^2}}$
5. $\int_{1,4}^{2,1} \frac{dx}{\sqrt{3x^2-1}}$
6. $\int_{1,2}^{2,4} \frac{dx}{\sqrt{0,5+x^2}}$
7. $\int_{0,4}^{1,2} \frac{dx}{\sqrt{3+x^2}}$
8. $\int_{0,6}^{1,5} \frac{dx}{\sqrt{1+2x^2}}$
9. $\int_{2}^{3,5} \frac{dx}{\sqrt{x^2-1}}$
10. $\int_{1,2}^{2} \frac{\lg(x+2)}{x} dx$
11. $\int_{1,6}^{2,4} (x+1)\sin x dx$
12. $\int_{0,2}^{1} \frac{tgx^2}{x^2+1} dx$
13. $\int_{0,6}^{1,4} \frac{\cos x}{x+1} dx$
14. $\int_{0,4}^{1,2} \sqrt{x\cos x^2 dx}$
15. $\int_{0,8}^{1,2} \frac{\sin 2x}{x^2} dx$
16. $\int_{0,8}^{1,6} \frac{\lg(x^2+1)}{x} dx$
17. $\int_{0,4}^{1,2} \frac{\cos x}{x+2} dx$
18. $\int_{0,4}^{1,2} (2x+0,5)\sin x dx$
19. $\int_{0,4}^{0,8} \frac{tg(x^2+0,5)}{1+2x^2}$
20. $\int_{0,18}^{0,98} \frac{\sin x}{x+1} dx$

РОЗДІЛ 17. ЧИСЕЛЬНІ МЕТОДИ РОЗВ'ЯЗАННЯ АЛГЕБРАЇЧНИХ РІВНЯНЬ

17.1. Метод половинного ділення (дихотомія)

Якщо корінь відокремлений, то для знаходження кореня рівняння ділимо відрізок [*a*, *b*] навпіл: *c* = (*a* + *b*)/2. Обов'язково один з відрізків [*a*, *c*] або [*c*, *b*] задовольняє умовам теореми і містить корінь рівняння, але має довжину в два рази менше, ніж початковий. В результаті застосовуємо цей прийом до нового відрізку, і отримуємо послідовність вкладених відрізків, довжина яких прямує до 0, а кінці цих відрізків прямують до кореня заданого рівняння.

Наведемо фрагмент програми реалізації методу половинного ділення.

```
#include <iostream>
#include <cmath>
using namespace std;

#include <conio.h>

double func(double x)
{
 return x*x + 5*x + 6;
}

// Розв'язання рівняння методом половинного ділення
// (знаходження тільки одного кореня)

void main()
{
 double a,b,c,fa,fb,fc,eps;

 a = -1000;
 b = -2.5;

 eps = 0.00000001;

 fa = func(a);
 fb = func(b);
  if(fa*fb>0)
 {
  cout<<"Невірні значення a та b";
  getch();
  exit(0);
 }

 c = (a+b)/2;
```

```
 fc = func(c);

 while(fabs(a-b)>eps&&fabs(fc)>eps)
 {
  if(fc*fa<=0)
    {
     fb = fc;
     b = c;
    }
    else
    {
     fa = fc;
     a = c;
    }
   c = (a+b)/2;
   fc = func(c);
 }
  cout<<"Корінь рівняння "<<c;
  getch();
}
```

Таким чином даний метод на заданому відрізку має $log_2(b-a)$ ітерацій, де a та b, відповідно, початок та кінець відрізку на якому знаходиться єдиний корінь рівняння. Слід зауважити, що у даному прикладі знайдено один корінь квадратного рівняння $x^2 + 5x + 6 = 0$, що знаходиться на проміжку [-1000; -2,5].

Результат виконання програми:

```
Корінь рівняння       -3
```

17.2. Метод Ньютона (метод дотичних)

Нехай корінь рівняння $f(x)= 0$ відокремлений на відрізку $[a, b]$, причому перша і друга похідна визначені, безперервні і зберігають певний знак на цьому відрізку. Тоді ітераційний процес Ньютона:

$$x_{n+1} = x_n - f(x_n) / f'(x_n)$$

сходиться для будь-якого початкового наближення x_0, що належить відрізку $[a, b]$. Причому повинна виконуватися нерівність:

$$f(x_0)*f''(x_0)>0.$$

Оскільки $f(a)*f(b)<0$, то у якості x_0 зручно вибрати точку a або b залежно від того більше або менше нуля друга похідна функції f на відрізку $[a, b]$. Аналіз показує, що за будь-яких умов один з кінців відрізків можна взяти за початкове наближення процесу.

Наведемо фрагмент програми реалізації методу Ньютона (рівняння $x^2 + 5x + 6 = 0$, проміжок пошуку кореня [-1000; -2,5]).

```
#include <iostream>
#include <cmath>
using namespace std;

#include <conio.h>

double func(double x)
{
 return x*x + 5*x + 6;
}

double difer1(double x,double h)
{
 return (func(x+h)-func(x-h))/(2*h);
}

double difer2(double x,double h)
{
 return (func(x+h)-2*func(x)+func(x-h))/(h*h);
}

// Розв'язання рівняння методом Ньютона
// (методом дотичних)
// (знаходження тільки одного кореня)

void main()
{
 double a,b,fa,fa2,fb,h,eps;
 double x,y,x1,y1;
 long l,i,n;

 n = 1000;
 a = -1000;
 b = -2.5;

 eps = 0.00000001;
 h = 0.00001;

 fa = func(a);
 fb = func(b);
 if(fa*fb>0)
     {
       cout<<"Невірні значення a та b";
       getch();
       exit(0);
     }
```

```
 fa2 = difer2(a,h);

 //Вибір нерухомого кінця відрізку

 if(fa*fa2>0)
             {
              x = a;
              l = 1;
             }
 else
             {
              x = b;
              l = -1;
             }

for(i=1;i<n;i++)
 {
  y = func(x);
  y1 = difer1(x,h);

  if(fabs(y1)<eps&&fabs(y)>eps)
   {
    cout<<"\nПерша похідна функції повинна бути
             знакопостійна!";
    break;
   }
   x1 = x - y/y1;
   if((l==1&&x>x1)||(l==-1&&x<x1))
   {
    cout<<"\nПорушені умови збіжності методу
             Ньютона!";
    cout<<"\nКорінь рівняння не знайдено!";
    break;
   }

   if(fabs(x-x1)<eps)
   {
    cout<<"\nКорінь рівняння "<<x1;
    break;
   }
   else x = x1;

 }//end for
```

```
 if(i==n)cout<<"\nКорінь рівняння не знайдено!";

 getch();
}
```

Результат виконання програми:

```
Корінь рівняння       -3
```

Методи, розглянуті вище, призначені для пошуку лише одного кореня рівняння. Якщо їх декілька, необхідно знаходити такі проміжки пошуку, де буде знаходитись один корінь. Для пошуку всіх дійсних коренів рівняння використовується метод Рибакова.

17.3. Метод Рибакова

Метод Рибакова призначений для визначення всіх дійсних коренів рівняння $f(x) = 0$ на відрізку $[a, b]$. Для збіжності цього методу достатньо, щоб $f(x)$ була безперервною, а похідна цієї функції обмежена на всьому відрізку $[a, b]$. Потрібно визначити $M > max|f'(x)|$ на цьому відрізку. Відмітимо, що завищення M не порушує збіжності методу, а тільки уповільнює її. Як початкове наближення вибирається лівий кінець відрізку – $x_0 = a$. Для кожного n обчислюють чергове наближення за формулою:

$$x_{n+1} = x_n + |f(x_n)|/M.$$

При цьому перевіряється умова $x_{n+1}<b$. Якщо вона не виконана, то вважається, що всі корені знайдені, інакше продовжують обчислення, перевіряючи нерівність: $|f(x_n)|<e$.

Якщо нерівність виконана, то вважають, що x_n черговий корінь рівняння $f(x)$. Далі вважають $x_{n+1} = x_n + e$ та продовжують процес знаходження коренів за формулою. Таким чином знаходяться всі дійсні корені без урахування їх кратності. Цей метод нагадує модифікований метод Ньютона. Слід відмітити, що збіжність методу Рибакова достатньо повільна.

Наведемо фрагмент програми реалізації методу Рибакова на прикладі рівняння $x^2 + 5x + 6 = 0$ (проміжок пошуку коренів [-1000; 1000]).

```
#include <iostream>
#include <cmath>
using namespace std;

#include <conio.h>

double func(double x)
{
 return x*x + 5*x + 6;
}
```

```
double difer(double x,double h)
{
 return (func(x+h)-func(x-h))/(2*h);
}

// Розв'язання рівняння методом Рибакова
// (знаходження всіх дійсних коренів)

void main()
{
 double a,b,h,x,y,y1,m,eps,difa,difb;

 a = -1000;
 b = 1000;

 eps = 0.00000001;
 h = eps;

 // Знаходження максимального модулю похідної
 // на заданому відрізку

 difa = fabs(difer(a,h));
 difb = fabs(difer(b,h));

 m = (difa>difb)?difa:difb;
 m++;
 ////////////////////////////

 x = a;
 do
 {
  y = func(x);
  y1 = difer(x,h);
  if(fabs(y1)>=m)
  {
   cout<<"\nПерша похідна функції повинна бути менше
            m по модулю !";
   getch();
   exit(0);
  }

  if(fabs(y)<eps)
  {
   cout<<"\nКорінь рівняння x="<<x;
   x+=10*eps;
```

```
    y=func(x);
   }

   x+=fabs(y/m);

  }while(x<b);

  getch();
}
```

Важливим у даному фрагменті є вибір максимальної похідної на заданому відрізку (величина *m*). У якості варіанта запропоновано знайти максимальне значення похідної по абсолютній величині на кінцях відрізку.

Результат виконання програми:

```
Корінь рівняння       x = -3
Корень рівняння       x = -2
```

Контрольні питання

1. Які ви знаєте чисельні методи розв'язання алгебраїчних рівнянь?
2. Розкрийте сутність методу половинного ділення розв'язання алгебраїчних рівнянь. Наведіть фрагмент програми.
3. Розкрийте сутність методу Ньютона розв'язання алгебраїчних рівнянь. Наведіть фрагмент програми.
4. Розкрийте сутність методу Рибакова розв'язання алгебраїчних рівнянь. Наведіть фрагмент програми.

Завдання

Скласти програму розв'язання алгебраїчних рівнянь за допомогою чисельних методів. Проміжки знаходження кореня рівняння та величину помилки обрати самостійно.

Варіанти індивідуальних завдань

1. Розв'язати рівняння $x^2 - 5x + 6 = 0$, використовуючи метод половинного ділення.
2. Розв'язати рівняння $2x^2 - 5x + 1 = 0$, використовуючи метод Ньютона.
3. Розв'язати рівняння $x^6 - 9x^3 + 8 = 0$, використовуючи метод Рибакова.
4. Розв'язати рівняння $3x^2 + 17x - 14 = 0$, використовуючи метод половинного ділення.
5. Розв'язати рівняння $\frac{2}{x-1} + \frac{5}{x+2} = \frac{13}{(x-1)(x+2)}$, використовуючи метод Ньютона.

6. Розв’язати рівняння $\frac{1}{x(x+2)}-\frac{1}{(x+1)^2}=\frac{1}{12}$, використовуючи метод Рибакова.

7. Розв’язати рівняння $(x^2-4x)^2-6(x-2)^2=16$, використовуючи метод Рибакова.

8. Розв’язати рівняння $x^3+8=0$, використовуючи метод половинного ділення.

9. Розв’язати рівняння $x^2-4=0$, використовуючи метод Ньютона.

10. Розв’язати рівняння $5x^4-2x^3-5x^2+2x=0$, використовуючи метод Рибакова.

11. Розв’язати рівняння $4x^2-8x+3=0$, використовуючи метод половинного ділення.

12. Розв’язати рівняння $2x^2-7x+3=0$, використовуючи метод Ньютона.

13. Розв’язати рівняння $5x^3-2x^2+5x-2=0$, використовуючи метод Рибакова.

14. Розв’язати рівняння $x^3+x^2-x-1=0$, використовуючи метод половинного ділення.

15. Розв’язати рівняння $x^3+x^2+x+1=0$, використовуючи метод Ньютона.

16. Розв’язати рівняння $x+\sqrt{x}-6=0$, використовуючи метод Рибакова.

17. Розв’язати рівняння $x-5\sqrt{x}+6=0$, використовуючи метод Рибакова.

18. Розв’язати рівняння $8-x^3=0$, використовуючи метод половинного ділення.

19. Розв’язати рівняння $x^2+x-12=0$, використовуючи метод Ньютона.

20. Розв’язати рівняння $x^4-2x^2+1=0$, використовуючи метод Рибакова.

21. Розв’язати рівняння $x^2+5x-6=0$, використовуючи метод половинного ділення.

22. Розв’язати рівняння $3x^2+2x-1=0$, використовуючи метод Ньютона.

23. Розв’язати рівняння $9x^3+9x^2-x-1=0$, використовуючи метод Рибакова.

24. Розв’язати рівняння $x^4-16=0$, використовуючи метод половинного ділення.

25. Розв’язати рівняння $2x^3-x^2-10x+1=0$, використовуючи метод Рибакова.

РОЗДІЛ 18. ЧИСЕЛЬНІ МЕТОДИ РОЗВ'ЯЗАННЯ СИСТЕМИ ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ

18.1. Визначник. Дії над матрицями. Обчислення оберненої матриці

Поняття визначника виникло у зв'язку з проблемою знаходження формул для невідомих в системі лінійних рівнянь. Розглянемо систему n рівнянь з n невідомими.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1j} + \ldots + a_{1n}x_n = c_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2j} + \ldots + a_{2n}x_n = c_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{ij} + \ldots + a_{in}x_n = c_i \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nj} + \ldots + a_{nn}x_n = c_n \end{cases}$$

де a_{ij} – коефіцієнти системи;
$i = 1, n$ – номер рівняння;
$j = 1, n$ – номер невідомого.

Коефіцієнти системи являють собою квадратну матрицю порядку n та вміщують n^2 елементів.

Визначником (детермінантом) даної системи є число Δ.

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{vmatrix}$$, де a_{ij} – елементи визначника.

Визначник складається тільки з квадратної матриці.

Розглянемо властивості визначника n-го порядку.

1. Визначник не змінюється при транспонуванні матриці (обертання матриці навколо головної діагоналі).

2. Якщо у визначника всі елементи деякого рядка або стовпця дорівнюють нулю, то визначник дорівнює нулю.

3. При перестановці двох рядків або стовпців визначник змінює знак на протилежний. Наприклад, при перестановці першого та другого рядків отримаємо:

$$\begin{vmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{vmatrix} = - \begin{vmatrix} a_{21} & a_{22} & \ldots & a_{2n} \\ a_{11} & a_{12} & \ldots & a_{1n} \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{vmatrix}$$

4. Якщо визначник має два однакових рядки або стовпці, то він дорівнює нулю.

5. Загальний множник усіх елементів деякого рядка можна виносити за знак визначника:

$$\begin{vmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \ldots & \lambda a_{2n} \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{vmatrix} = \lambda \begin{vmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{vmatrix}$$

6. Якщо усі елементи одного рядка (стовпця) пропорційні відповідним елементам другого рядка (стовпця), то визначник дорівнює нулю.

$$\begin{vmatrix} a_{11} & a_{12} & ... & a_{1n} \\ ca_{21} & ca_{22} & ... & ca_{2n} \\ a_{n1} & a_{n2} & ... & a_{nn} \end{vmatrix} = c \begin{vmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ a_{n1} & a_{n2} & ... & a_{nn} \end{vmatrix} = 0, c = const$$

7. Якщо до елементів одного із рядків (стовпців) додати відповідні елементи другого рядка (стовпця) помножені на одне й теж число, відмінне від нуля, то визначник не змінюється.

Властивості визначника використовуються при елементарних перетвореннях для їх обчислення.

Є багато способів розрахунку визначника *n*-го порядку. Найоптимальнішим засобом є приведення матриці визначника до трикутного виду (з нижнім або верхнім розташуванням нульових елементів). У цьому випадку визначник дорівнює добутку елементів, що стоять на головній діагоналі.

Авторами запропоновано універсальну функцію розрахунку визначника n-го порядку, фрагмент якої приведено нижче.

```
#include <iostream>
#include <cmath>
using namespace std;

# define delta 0.00000001
// ФУНКЦІЯ ЗНАХОДЖЕННЯ ВИЗНАЧНИКА ПОРЯДКУ N

//Перетворення нижнього трикутника визначника в 0
double mopred(double*k, int n)
{
 int i,j,xx,r,kol_el;
 double p,t,vv;
 kol_el = n*n;
 double*kk = new double[kol_el];

 //Перезапис елементів масиву
 for(i=0;i<kol_el;i++)
       kk[i] = k[i];

 for(i=n-1;i>=0;i--)
     for(j=0;j<i;j++)
      {
       if (fabs(kk[i*n+j+1])<delta)
          {
            // Перестановка стовпців якщо елемент
            // праворуч нульовий
            for(xx=0;xx<=i;xx++)
```

```
                {
                 vv = kk[xx*n+j];
                 kk[xx*n+j]=kk[xx*n+j+1];
                 kk[xx*n+j+1] = -vv;
               }
            continue;
            }//endif

          //Якщо поточний елемент нульовий або отримався
          //завдяки перестановці стовпців,
          //переходимо до наступного елемента
          if (fabs(kk[i*n+j])<delta)continue;
          //знаходимо результат від ділення двох сусідніх
..        //ненульових елементів
           t = kk[i*n+j]/kk[i*n+j+1];
          //перетворюємо поточний елемент в 0 -
          //використовуючи властивості визначника
          //множимо поточний стовпчик на число(результат
          //від ділення) та додаємо наступний стовпчик.
          //при цьому визначник не змінюється
            for(r=0;r<=i;r++)
              kk[r*n+j]=kk[r*n+j]-kk[r*n+j+1]*t;
        }

        p = 1;
        for(i=0;i<n;i++)
             p*=kk[i*n+i];

 delete[]kk;

 return p;
}
```

У даній функції **mopred**() відбувається перевірка нижнього трикутника матриці. Йде перевірка елементів по кожному рядку, починаючи з останнього. Алгоритм побудований наступним чином:

1. Якщо нульові елементи вже мають місце, вони будуть здвигатися вліво (за рахунок перестановки стовпців матриці).

2. Перераховуються тільки відмінні від нуля сусідні елементи, таким чином, щоб елемент, з яким іде робота, у результаті арифметичних перетворень дорівнював би нулю (використовуючи сьому властивість визначника).

Таким чином, якщо нульові елементи будуть знаходитись на головній діагоналі, або просто мати місце, їх буде здвинуто вліво, крім того стовпець матриці буде перераховуватись таким чином, щоб елемент, з яким відбувається робота, дорівнював би нулю. Для того щоб не псувати вхідні данні (матриця **k**), все записується у матрицю **kk**.

Увага! Для універсальності роботи функцій (передача довільних параметрів) всі матриці представлені у вигляді масивів. Так у кожну функцію передається вказівка на одномірний масив.

Продемонструємо роботу функції на конкретному прикладі (фрагмент програми).

```
……….
void main()
{
  double k[9]={
         1, 0, 8,
          2, 5, 9,
          3, 6, 0
         };

  cout<<"\nopred="<<mopred(k, 3);
}
……….
```

У результаті виконання програми визначник матриці $\begin{pmatrix} 1 & 0 & 8 \\ 2 & 5 & 9 \\ 3 & 6 & 0 \end{pmatrix}$ буде дорівнювати -78.

```
opred = -78
```

Лінійні дії над матрицями. Множення матриць. Обернена матриця.

Під лінійними діями розуміють додавання і віднімання матриць, множення матриць на скаляр (const). Операція + (–) визначається тільки для матриць однакових розмірів. Сумою (різницею) двох матриць A і B розміром $m_x n$ називається матриця C такого ж розміру у якій елементи обчислюються за формулою $c_{ij} = a_{ij} \pm b_{ij}$. Наприклад $C = A - B$, де

$$A\begin{pmatrix} 2 & 3 \\ 7 & 8 \\ 5 & 6 \end{pmatrix}; B\begin{pmatrix} -2 & 3 \\ -5 & 6 \\ 7 & 0 \end{pmatrix}$$

$$C = A - B = \begin{pmatrix} 4 & 0 \\ 12 & 2 \\ -2 & 6 \end{pmatrix}$$

Нульова матриця при + (–) матриць виконує роль звичайного нуля при + (–) чисел. $A \pm 0 = A$.

Добутком матриці A на λ (λ = const) називається матриця B, яка отримується з даної помноженням усіх її елементів на скаляр λ.

$$B = A \cdot \lambda = \lambda \cdot A = \lambda\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} \lambda a_{11} & \lambda a_{12} \\ \lambda a_{21} & \lambda a_{22} \end{pmatrix}$$

Помноження матриць відбувається тільки тоді, коли матриця A погоджена

з матрицею B (кількість стовпців матриці A дорівнює кількості рядків матриці B).

Тільки квадратні матриці одного порядку є взаємопогоджені. Правило погодження матриць називається правило "рядок на стовбець". Якщо для матриць A і B знайдені добутки $A*B$ і $B*A$ то це не значить, що $A*B = B*A$. При помноженні квадратної матриці на нульову матрицю отримують нульову матрицю.

$$A \cdot 0 = 0 \cdot A = 0$$

$$A \cdot B = \begin{pmatrix} 2 & 1 & 3 \\ -1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 2 & 4 \\ -1 & 6 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 2 \cdot 2 + 1 \cdot (-1) + 3 \cdot 3 & 2 \cdot 4 + 1 \cdot 6 + 3 \cdot 1 \\ -1 \cdot 2 + 0 \cdot (-1) + 2 \cdot 3 & -1 \cdot 4 + 0 \cdot 6 + 2 \cdot 1 \end{pmatrix} = \begin{pmatrix} 12 & 17 \\ 4 & -2 \end{pmatrix} = C$$

Якщо A і B квадратні матриці одного порядку з визначниками $|A|$ і $|B|$, то для матриці $C = A*B$, її визначник $|C| = |A*B| = |A| * |B|$.

Розглянемо квадратну матрицю n-го порядку. Квадратна матриця A^{-1} називається оберненою для квадратної матриці A того ж порядку, якщо виконується рівність $A^{-1} \cdot A = A \cdot A^{-1} = E$.

Для того, щоб квадратна матриця A мала обернену, необхідно і достатньо щоб матриця A була невиродженою, тобто щоб її визначник був відмінний від нуля. $(|A| \neq 0)$.

Формула оберненою матриці.

$$A^{-1} = \frac{1}{|A|} \cdot B = \frac{1}{|A|} \cdot \begin{pmatrix} A_{11} & A_{21} & A_{31} \\ A_{12} & A_{22} & A_{32} \\ A_{13} & A_{23} & A_{33} \end{pmatrix},$$

де матриця B – транспонована матриця A, у якій елементи замінені алгебраїчними доповненнями.

Необхідною перевіркою правильності знаходження оберненої матриці є рівність $A^{-1} \cdot A = E$.

З точки зору обчислення оберненої матриці – обчислення за допомогою приведеної формули є неоптимальним варіантом, так як при цьому необхідно обчислити n^2 алгебраїчних доповнень (визначників порядку n-1), а це займає досить багато часу. Одним з самих швидких методів обчислення оберненої матриці є метод Жордана-Гауса, який зводиться до вирішення системи лінійних рівнянь.

Позначимо елементи оберненої матриці через α_{lm}. Тоді співвідношення $AA^{-1} = E$ можна записати так:

$$\sum_{k=1}^{n} a_{ik} \alpha_{kl} = \delta_{il}, \; l \le i, \; l \le n .$$

Якщо розглядати l-й стовпець оберненої матриці як вектор, то він є вирішенням лінійної системи з матрицею A і спеціальною правою частиною (у якій на l-м місці знаходиться одиниця, а на останніх – нулі).

Таким чином, для обернення матриці треба розв’язати n систем лінійних рівнянь з однаковою матрицею A і різними правими частинами. Приведення матриці A до трикутної форми робиться при цьому тільки один раз.

Цікаво відзначити, що обернення матриці зводиться до вирішення *n* систем лінійних рівнянь, і вимагає лише втричі більше дій ніж вирішення однієї системи рівнянь. Це пояснюється тим, що при вирішенні лінійної системи велика частина обчислень пов'язана з приведенням матриці до трикутного виду, що при оберненні матриці робиться тільки один раз. Зворотний хід і перетворення правих частин виконуються набагато швидше. Тому, якщо потрібно кілька разів розв’язати лінійну систему з однією і тією ж матрицею, то вигідно привести матрицю до трикутної форми лише один раз.

Наведемо код функцій множення матриць та оберненої матриці методом Жордана-Гауса.

```
#include <iostream>
#include <cmath>
using namespace std;

# define delta 0.00000001

// Ф-я множення 2-х матриць
// Вход:
/*
    m1  –  масив-шнурок представлення 1-ї матриці
           розмірності m*n
    m2  –  масив шнурок представлення 2-ї матриці
           розмірності n*l
    Обов’язкова умова 2-х вхідних матриць (кількість
    стовбців 1-ї = кількості рядків 2-ї)
    m, n, l – розмірності відповідних матриць
*/
// Вихід m3 – отримана матриця (у вигляді масиву)
// розмірністю m*l елементів

void mumnog(double*m1, double*m2,
             int m, int n, int l, double*m3)
{
  int i,j,k;
  for(i=0;i<m;i++)
      for(j=0;j<l;j++)
       {
       m3[i*l+j] = 0;
        for(k=0;k<n;k++)
            m3[i*l+j]+=m1[i*n+k]*m2[k*l+j];
      }
}
```

```
/*
   Ф-я знаходження оберненої матриці порядку n методом
    Гаусса
   k - масив входу (матриця A)
  bb - масив виходу (матриця A^-1)
*/
bool mObr_Gauss(double*k,double*bb,int n)
{
 int i,j,tt,xx,r,kol_el;
 double t,vv;
 kol_el = n*n;

 //Якщо визначник 0 - оберненої матриці не існує.
 if(fabs(mopred(k,n))<delta)return false;

 double*aa = new double[kol_el];
 double*kk = new double[kol_el];

   //Перегонка масивів
       for(i=0;i<kol_el;i++)
       {
        aa[i] = 0;
        kk[i] = k[i];
       }

 //Заповнення 1 по головній діагоналі -
 //aa - одинична матриця
 for(i=0;i<n;i++)aa[i*n+i]=1;

   //Приведення нижнього трикутника до нульових
   //коефіцієнтів (методом Гауса)
   //Маніпуляції виконуються як з поточною,
   //так і з одиничною матрицею
     for(j=0;j<n;j++)
      for(i=n-1;i>j;i--)
     {
      if (fabs(kk[(i-1)*n+j])<delta)
         {
           for(xx=j;xx<n;xx++)
              {
               vv = kk[i*n+xx];
               kk[i*n+xx]=kk[(i-1)*n+xx];
               kk[(i-1)*n+xx] = vv;
             }
```

```
                for(tt=0;tt<n;tt++)
                {
                 vv = aa[i+tt*n];
                 aa[i+tt*n] = aa[i+tt*n-1];
                 aa[i+tt*n-1] = vv;
                }
              continue;
             }//endif
          if (fabs(kk[i*n+j])<delta)continue;
        t = kk[i*n+j]/kk[(i-1)*n+j];

            for(r=j;r<=n;r++)
              kk[i*n+r]=kk[i*n+r]-kk[(i-1)*n+r]*t;

            for(tt=0;tt<n;tt++)
              aa[i+tt*n]=aa[i+tt*n]-aa[i+tt*n-1]*t;
          }

        //Для кожного стовбця зміненої одиничної матриці
        //знаходиться розв'язок
        //Таким чином отримуємо обернену матрицю,
        //тобто коли A->E, E->A^(-1)
        for (i=n-1;i>=0;i--)
        {
           for (j=n-1;j>i;j--)
             for(tt=0;tt<n;tt++)
              aa[i+tt*n]-=kk[i*n+j]*aa[j+tt*n];

             for(tt=0;tt<n;tt++)
              aa[i+tt*n]/=kk[i*n+i];
        }

       //Запис вихідного масиву(транспонований вид)
       for(i=0;i<kol_el;i++)bb[i] = aa[(i%n)*n + i/n];

       delete[]aa;
       delete[]kk;

       return true;
      }
```

Широкого поширення набули прямі чисельні методи розв'язання системи лінійних алгебраїчних рівнянь: оберненої матриці, Крамера, Гауса. Розв'яжемо систему лінійних рівнянь:

$$\begin{cases} x_1 + 8x_3 = 6 \\ 2x_1 + 5x_2 + 9x_3 = 4 \\ 3x_1 + 6x_2 = 9 \end{cases}$$

кожним з цих методів.

18.2. Метод оберненої матриці.

Одним з прямих методів розв'язання системи лінійних рівнянь є метод оберненої матриці. Розглянемо систему *n* лінійних рівнянь з *n* невідомими. Складемо матрицю системи із коефіцієнтів при невідомих і матриці-стовбці невідомих і вільних членів:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix},\ X = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix},\ C = \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{pmatrix}.$$

Представимо задану систему в еквівалентному матричному вигляді.

$A \cdot X = C$

Помножимо обидві частини даного рівняння на обернену матрицю (A^{-1}), вважаючи, що матриця системи *A* невироджена $(|A| \neq 0)$.

$A^{-1} \cdot A \cdot X = A^{-1} \cdot C$, $E \cdot X = A^{-1} \cdot C$, $X = A^{-1} \cdot C$ – розв'язок матричного рівняння і початкової системи.

Приведемо фрагмент вирішення системи лінійних рівнянь матричним способом.

```
.....................................................................
void main()
{
  double k[9]={
         1, 0, 8,
         2, 5, 9,
         3, 6, 0
         };
  double cc[9],dd[3];
  double aaa[3]={6, 4, 9};

  mObr_Gauss(k,cc,3);
  mumnog(cc, aaa, 3, 3, 1, dd);

  cout<<"\n\n"<<cc[0]<<"\t"<<cc[1]<<"\t"<<cc[2];
  cout<<"\n"<<cc[3]<<"\t"<<cc[4]<<"\t"<<cc[5];
  cout<<"\n"<<cc[6]<<"\t"<<cc[7]<<"\t"<<cc[8];
  cout<<"\n\nx1="<<dd[0]<<"\tx2="<<dd[1]
      <<"\tx3="<<dd[2];
}
.....................................................................
```

У результаті виконання програми, використовуючи функції **mObr_Gauss()** для обчислення оберненої матриці, та функції **mumnog()** для множення матриць знайдено обернену матрицю (**cc**) та корені рівняння (**dd**):

0.692308 -0.615385 0.512821
-0.346154 0.307692 -0.0897436
0.0384615 0.0769231 -0.0641026

x1 = 6.30769 x2 = -1.65385 x3 = -0.0384615

18.3. Метод Крамера

Розглянемо систему *n* лінійних рівнянь з *n* невідомими.

$\Delta = \begin{vmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nn} \end{vmatrix}$ будемо називати визначником системи.

Нехай $\Delta \neq 0$:

Якщо визначник системи Δ відмінний від нуля то ця система має лише один розв'язок, який знаходять по наступним формулам:

$$x_1 = \frac{\Delta x_1}{\Delta}; x_2 = \frac{\Delta x_2}{\Delta}; ...; x_n = \frac{\Delta x_i}{\Delta}; ...; x_n = \frac{\Delta x_n}{\Delta}$$

Δx_i – це визначник системи, у якому необхідно замінити коефіцієнти, що стоять попереду невідомого x_i на стовбець вільних членів.

$$\Delta x_1 = \begin{vmatrix} c_1 & a_{12} & ... & a_{1n} \\ c_2 & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ c_n & a_{n2} & ... & a_{nn} \end{vmatrix}$$

Наведемо фрагмент реалізації системи лінійних рівнянь з 3-ма невідомими методом Крамера.

```
.....................................................................
void main()
{
 double d,dx1,dx2,dx3,x1,x2,x3;

  double k[9]={
               1,0,8,
               2,5,9,
               3,6,0
               };
  /*
  double aaa[3]={
                 6,
                 4,
```

```
                9
                };
 */
double aax1[9]={
                 6,0,8,
                 4,5,9,
                 9,6,0
                };

double aax2[9]={
                 1,6,8,
                 2,4,9,
                 3,9,0
                };
double aax3[9]={
                 1,0,6,
                 2,5,4,
                 3,6,9
                };

d = mopred(k, 3);
dx1 = mopred(aax1, 3);
dx2 = mopred(aax2, 3);
dx3 = mopred(aax3, 3);

x1 = dx1/d;
x2 = dx2/d;
x3 = dx3/d;

cout<<"\nx1="<<x1<<"\tx2="<<x2<<"\tx3="<<x3;
}
```

.....

У результаті виконання програми, будуть виведені наступні результати:

x1 = 6.30769 x2 = -1.65385 x3 = -0.0384615

18.4. Метод Гауса

Метод Гауса – це метод послідовного виключення невідомих. Він є самим швидким методом розв'язку системи лінійних рівнянь порівняно з методами оберненої матриці та методом Крамера. Нехай задана система *n* лінійних рівнянь з *n* невідомими. Складемо розширену матрицю системи.

$$B = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix}$$

Шляхом елементарних перетворень із матриці B побудуємо трикутну матрицю. Ця система має лише один розв'язок (визначник не повинен дорівнювати нулю). Невідомі знаходяться шляхом розв'язку у зворотному порядку.

Нижче приведемо фрагмент програми, за допомогою якого вирішується система лінійних рівнянь використовуючи метод Гауса.

```
#include <iostream>
#include <cmath>
using namespace std;

# define delta 0.00000001

// aaa - массив свободных членов
// bb - выходной массив

bool mGauss(double*k,double*aaa,double*bb,int n)
{
 int i,j,xx,r,kol_el;
 double t,vv;
 kol_el = n*n;

 //Якщо визначник 0 - система має безліч коренів.
 if(fabs(mopred(k,n))<delta)return false;

 double*aa = new double[n];
 double*kk = new double[kol_el];

   //Перегонка масивів
    for(i=0;i<n;i++)
        aa[i] = aaa[i];
    for(i=0;i<kol_el;i++)
        kk[i] = k[i];

   // Перетворення коефіцієнтів рівняння до
   // трикутного виду
   // нижній трикутник 0
   // переставляються та продвигаються рядки (також
   // як і в методиці знаходження визначника,
   // тільки стовбці замінені на рядки, так як
   // кожному рівнянню відповідає визначений рядок
```

```
   // матриці

     for(j=0;j<n;j++)
      for(i=n-1;i>j;i--)
     {
      if (fabs(kk[(i-1)*n+j])<delta)
         {
           for(xx=j;xx<n;xx++)
             {
             vv = kk[i*n+xx];
             kk[i*n+xx]=kk[(i-1)*n+xx];
             kk[(i-1)*n+xx] = vv;
            }

             vv = aa[i];aa[i] = aa[i-1];aa[i-1] = vv;
          continue;
         }//endif
      if (fabs(kk[i*n+j])<delta)continue;
     t = kk[i*n+j]/kk[(i-1)*n+j];

        for(r=j;r<n;r++)
          kk[i*n+r]=kk[i*n+r]-kk[(i-1)*n+r]*t;
          aa[i]=aa[i]-aa[i-1]*t;
     }

   /*  Знаходження коренів рівняння обратним ходом
       (знизу вверх)
       по зміні матриці невідомих та масиву вихідних
       членів
   */

   for (i=n-1;i>=0;i--)
   {
      for (j=n-1;j>i;j--)
         aa[i]-=kk[i*n+j]*aa[j];
      aa[i]/=kk[i*n+i];

   }

   for(i=0;i<n;i++)bb[i] = aa[i];

 delete[]aa;
 delete[]kk;

 return true;
}
```

```
void main()
{
  double k[9]={
         1, 0, 8,
         2, 5, 9,
         3, 6, 0
         };
  double bb[3];
  double aa[3]={6, 4, 9};
  mGauss(k,aa,bb,3);
  cout<<"\nx1="<<bb[0]<<"\x2="<<bb[1]
      <<"\tx3="<<bb[2];
}
```

У представленій функції **mGauss()** спочатку перевіряється визначник матриці коефіцієнтів рівняння (масив **k**). Якщо він не дорівнює нулю, матриця за допомогою алгоритму описаному вище (при отриманні визначника) – зводиться до трикутної форми (нижній трикутник заповнюється нулями). Відмінність полягає в тому, що операції здійснюються не над стовпчиками, а над рядками, так як кожен рядок характеризує відповідне рівняння. Крім того, розглядається ще й масив вільних членів (**aaa**). Після приведення матриці до трикутної форми – у зворотному порядку знаходяться корені рівняння. **Серед прямих методів розв'язку системи лінійних рівнянь метод Гауса є самим швидким та оптимальним.**

У результаті виконання програми одержимо корені рівняння на екрані дисплею:

x1 = 6.30769 x2 = -1.65385 x3 = -0.0384615

Контрольні питання

1. Які ви знаєте чисельні методи розв'язання системи лінійних алгебраїчних рівнянь?
2. Які дії можна робити над матрицями?
3. Яким чином обчислюється обернена матриця? Наведіть фрагмент програми.
4. Розкрийте сутність методу оберненої матриці розв'язання системи лінійних алгебраїчних рівнянь. Наведіть фрагмент програми.
5. Розкрийте сутність методу Крамера розв'язання системи лінійних алгебраїчних рівнянь. Наведіть фрагмент програми.
6. Розкрийте сутність методу Гауса розв'язання системи лінійних алгебраїчних рівнянь. Наведіть фрагмент програми.

Завдання

Скласти програму розв'язання системи лінійних алгебраїчних рівнянь за допомогою прямих чисельних методів: оберненої матриці, Крамера, Гауса. Виконати лінійні операції над матрицями.

Варіанти індивідуальних завдань

1. Розв'язати систему алгебраїчних рівнянь.

1. Розв'язати систему рівнянь методом оберненої матриці:

$$\begin{cases} -3x+2y-4z=15 \\ 4x-5y+3z=-27 \\ 5x+3y-4z=28 \end{cases}$$

2. Розв'язати систему рівнянь методом Крамера:

$$\begin{cases} 4x-y+2z=18 \\ 2x-3y+3z=8 \\ -4x-y-5z=-28 \end{cases}$$

3. Розв'язати систему рівнянь методом Гауса:

$$\begin{cases} 3x+2y+4z=-9 \\ 2x-2y+5z=2 \\ 4x-y+4z=-11 \end{cases}$$

4. Розв'язати систему рівнянь методом оберненої матриці:

$$\begin{cases} 4x+4y+z=-27 \\ -5x+5y+4z=-1 \\ 2x-y-3z=-5 \end{cases}$$

5. Розв'язати систему рівнянь методом Крамера:

$$\begin{cases} 4x+y-3z=-34 \\ -5x-2y+2z=41 \\ 2x-4y-3z=1 \end{cases}$$

6. Розв'язати систему рівнянь методом Гауса:

$$\begin{cases} 5x+4y+4z=-3 \\ -5x-3y+4z=-1 \\ 5x-2y-2z=-21 \end{cases}$$

7. Розв'язати систему рівнянь методом оберненої матриці:

$$\begin{cases} 4x+4y-4z=4 \\ -5x-y+3z=11 \\ -5x+3y-2z=21 \end{cases}$$

8. Розв’язати систему рівнянь методом Крамера:

$$\begin{cases} 2x-5y-5z=-1 \\ -5x-3y+z=26 \\ 5x-2y-2z=-13 \end{cases}$$

9. Розв’язати систему рівнянь методом Гауса:

$$\begin{cases} 3x+3y+3z=-27 \\ -2x-5y+5z=-2 \\ -2x+2y+z=-17 \end{cases}$$

10. Розв’язати систему рівнянь методом оберненої матриці:

$$\begin{cases} 3x-3y+4z=8 \\ 2x-3y-3z=33 \\ 3x-3y-2z=32 \end{cases}$$

11. Розв’язати систему рівнянь методом Крамера:

$$\begin{cases} 4x-2y-3z=-13 \\ 2x+5y+3z=-32 \\ -4x-2y-5z=25 \end{cases}$$

12. Розв’язати систему рівнянь методом Гауса:

$$\begin{cases} 3x-y+5z=-10 \\ 4x+5y+4z=-19 \\ 4x+4y-2z=14 \end{cases}$$

13. Розв’язати систему рівнянь методом оберненої матриці:

$$\begin{cases} 3x-4y-3z=21 \\ -5x+5y-2z=-44 \\ 3x-2y+z=23 \end{cases}$$

14. Розв’язати систему рівнянь методом Крамера:

$$\begin{cases} 5x+2y-3z=-37 \\ -5x+2y+z=21 \\ 2x+4y+z=-20 \end{cases}$$

15. Розв’язати систему рівнянь методом Гауса:

$$\begin{cases} 3x-2y+4z=31 \\ 4x+2y-2z=10 \\ -3x-5y+2z=1 \end{cases}$$

16. Розв’язати систему рівнянь методом оберненої матриці:

$$\begin{cases} -3x - y - 5z = 1 \\ -4x + 3y + 5z = 16 \\ 3x - 5y - 4z = -7 \end{cases}$$

17. Розв’язати систему рівнянь методом Крамера:

$$\begin{cases} 5x - 3y - 2z = -2 \\ 2x + 5y - 2z = 29 \\ 5x + 5y + z = 41 \end{cases}$$

18. Розв’язати систему рівнянь методом Гауса:

$$\begin{cases} -3x + 4y + z = 1 \\ -4x - y - 5z = 33 \\ 2x - 4y - 3z = 9 \end{cases}$$

19. Розв’язати систему рівнянь методом оберненої матриці:

$$\begin{cases} 3x - 2y + 3z = -15 \\ -3x - 3y - 4z = 5 \\ -4x - 4y - 3z = -5 \end{cases}$$

20. Розв’язати систему рівнянь методом Крамера:

$$\begin{cases} -3x + y - 5z = 2 \\ 4x + 5y - 5z = 7 \\ 2x - 3y + 2z = 12 \end{cases}$$

21. Розв’язати систему рівнянь методом Гауса:

$$\begin{cases} -5x - y - 4z = 44 \\ 2x + 3y - 4z = 13 \\ -2x + 5y + 2z = 5 \end{cases}$$

22. Розв’язати систему рівнянь методом оберненої матриці:

$$\begin{cases} 2x + 3y - 4z = -3 \\ 5x - 5y + 5z = 40 \\ 2x + y + 5z = 31 \end{cases}$$

23. Розв’язати систему рівнянь методом Крамера:

$$\begin{cases} -3x - 5y + 3z = 31 \\ -5x + 2y + 2z = 3 \\ -3x + 3y + 5z = -11 \end{cases}$$

24. Розв’язати систему рівнянь методом Гауса:

$$\begin{cases} -5x-2y-4z=-2 \\ -5x-4y-3z=1 \\ 4x+3y+4z=-2 \end{cases}$$

25. Розв’язати систему рівнянь методом оберненої матриці:

$$\begin{cases} 3x-5y+z=-17 \\ 2x-5y+3z=-6 \\ 3x-4y-2z=-25 \end{cases}$$

2. Обчислити добуток матриць.

1. $\begin{pmatrix} 3 & -2 \\ 5 & -4 \end{pmatrix} \cdot \begin{pmatrix} 3 & 4 \\ 2 & 5 \end{pmatrix}$

2. $\begin{pmatrix} 1 & -3 & 2 \\ 3 & -4 & 1 \\ 2 & -5 & 3 \end{pmatrix} \cdot \begin{pmatrix} 2 & 5 & 6 \\ 1 & 2 & 5 \\ 1 & 3 & 2 \end{pmatrix}$

3. $\begin{pmatrix} 5 & 8 & -4 \\ 6 & 9 & -5 \\ 4 & 7 & -3 \end{pmatrix} \cdot \begin{pmatrix} 3 & 2 & 5 \\ 4 & -1 & 3 \\ 9 & 6 & 5 \end{pmatrix}$

4. $\begin{pmatrix} 2 & -1 & 3 & -4 \\ 3 & -2 & 4 & -3 \\ 5 & -3 & -2 & 1 \\ 3 & -3 & -1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 7 & 8 & 6 & 9 \\ 5 & 7 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 2 & 1 & 1 & 2 \end{pmatrix}$

5. $\begin{pmatrix} 5 & 7 & -3 & -4 \\ 7 & 6 & -4 & -5 \\ 6 & 4 & -3 & -2 \\ 8 & 5 & -6 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{pmatrix}$

6. $\begin{pmatrix} 2 & -3 \\ 4 & -6 \end{pmatrix} \cdot \begin{pmatrix} 9 & -6 \\ 6 & -4 \end{pmatrix}$

7. $\begin{pmatrix} 5 & 2 & -2 & -3 \\ 6 & 4 & -3 & 5 \\ 9 & 2 & -3 & 4 \\ 7 & 6 & -4 & 7 \end{pmatrix} \cdot \begin{pmatrix} 2 & 2 & 2 & 2 \\ -1 & -5 & 3 & 11 \\ 16 & 24 & 8 & -8 \\ 8 & 16 & 0 & -16 \end{pmatrix}$

8. $\begin{pmatrix} 4 & 3 \\ 7 & 5 \end{pmatrix} \cdot \begin{pmatrix} -28 & 93 \\ 38 & -126 \end{pmatrix} \cdot \begin{pmatrix} 7 & 3 \\ 2 & 1 \end{pmatrix}$

9. $\begin{pmatrix} 0 & 2 & -1 \\ -2 & -1 & 2 \\ 3 & -2 & -1 \end{pmatrix} \cdot \begin{pmatrix} 70 & 34 & -107 \\ 52 & 26 & -68 \\ 101 & 50 & -140 \end{pmatrix} \cdot \begin{pmatrix} 27 & -18 & 10 \\ -46 & 31 & -17 \\ 3 & 2 & 1 \end{pmatrix}$

10. $\begin{pmatrix} 1 & 1 & 1 & -1 \\ -5 & -3 & -4 & 4 \\ 5 & 1 & 4 & -3 \\ -16 & -11 & -15 & 14 \end{pmatrix} \cdot \begin{pmatrix} 7 & -2 & 3 & 4 \\ 11 & 0 & 3 & 4 \\ 5 & 4 & 3 & 0 \\ 22 & 2 & 9 & 8 \end{pmatrix}$

3. Розв'язати матричні рівняння.

1. $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot X = \begin{pmatrix} 3 & 5 \\ 5 & 9 \end{pmatrix}$

2. $X \cdot \begin{pmatrix} 3 & -2 \\ 5 & -4 \end{pmatrix} = \begin{pmatrix} -1 & 2 \\ -5 & 6 \end{pmatrix}$

3. $\begin{pmatrix} 3 & -1 \\ 5 & -2 \end{pmatrix} \cdot X \cdot \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 14 & 16 \\ 9 & 10 \end{pmatrix}$

4. $\begin{pmatrix} 1 & 2 & -3 \\ 3 & 2 & -4 \\ 2 & -1 & 0 \end{pmatrix} \cdot X = \begin{pmatrix} 1 & -3 & 0 \\ 10 & 2 & 7 \\ 10 & 7 & 8 \end{pmatrix}$

5. $X \cdot \begin{pmatrix} 5 & 3 & 1 \\ 1 & -3 & -2 \\ -5 & 2 & 1 \end{pmatrix} = \begin{pmatrix} -8 & 3 & 0 \\ -5 & 9 & 0 \\ -2 & 15 & 0 \end{pmatrix}$

6. $\begin{pmatrix} 2 & -3 & 1 \\ 4 & -5 & 2 \\ 5 & -7 & 3 \end{pmatrix} \cdot X \cdot \begin{pmatrix} 9 & 7 & 6 \\ 1 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & -2 \\ 18 & 12 & 9 \\ 23 & 15 & 11 \end{pmatrix}$

7. $\begin{pmatrix} 2 & -3 \\ 4 & -6 \end{pmatrix} \cdot X = \begin{pmatrix} 2 & 3 \\ 4 & 6 \end{pmatrix}$

8. $X \cdot \begin{pmatrix} 3 & 6 \\ 4 & 8 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 9 & 18 \end{pmatrix}$

9. $\begin{pmatrix} 4 & 6 \\ 6 & 9 \end{pmatrix} \cdot X = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

10. $\begin{pmatrix} 3 & -1 & 2 \\ 4 & -3 & 3 \\ 1 & 3 & 0 \end{pmatrix} \cdot X = \begin{pmatrix} 3 & 9 & 7 \\ 1 & 11 & 7 \\ 7 & 5 & 7 \end{pmatrix}$

СПИСОК ЛІТЕРАТУРИ

1. Стивен Поттс, Тиноти С. Монк. Borland C++. (Минск, Попурри, 1996г. – 741 с.
2. Р. Уинер. Язык Турбо С. – М.: Мир, 1991г. - 384 с.
3. Грегори К. Использование Visual C++ 6. Учебное пособие – СПб: К.: Издательский дом "Вильямс", 1999. – 864 с.
4. Секунов Н.Ю. Самоучитель Visual C++ 6. – СПб.: БХВ – Санкт-Петербург, 1999. – 960 с.
5. Шилдт Г. Самоучитель C++, 3-е издание. – СПб.: BHV – Санкт-Петербург, 1998. – 688 с.
6. Фейсон Т. Объектно-ориентированное программирование на Borland C++ 4.5. – К.: "Диалектика", 1996. – 544 с.
7. Страуструп Б. Язык программирования C++. – СПб.: БИНОМ, 1999. – 991 с.
8. Страуструп Б. Дизайн и эволюция языка C++. – М.: ДМК, 2000. – 448 с.
9. Мейерс С. Эффективное использование C++. – М.: ДМК, 2000. – 240 с.
10. Павловская Т.А. C/C++. Программирование на языке высокого уровня. – СПб.: Питер, 2003. – 461 с.

ДОДАТКИ

Додаток 1. Таблиці ASCII-кодів Windows та DOS

Кодова таблиця системи Windows

Код	Симв.	Код	Симв.	Код	Симв.	Код	Симв.	Код	Симв.
Коди 0-127									
0	nul	26	eof	52	4	78	N	104	h
1	soh	27	esc	53	5	79	O	105	i
2	stx	28	fs	54	6	80	P	106	j
3	etx	29	qs	55	7	81	Q	107	k
4	eot	30	rs	56	8	82	R	108	l
5	enq	31	us	57	9	83	S	109	m
6	ack	32	sp	58	:	84	T	110	n
7	bel	33	!	59	;	85	U	111	o
8	bs	34	"	60	<	86	V	112	p
9	tab	35	#	61	=	87	W	113	q
10	lf	36	$	62	>	88	X	114	r
11	vt	37	%	63	?	89	Y	115	s
12	np	38	&	64	@	90	Z	116	t
13	cr	39	'	65	A	91	[	117	u
14	so	40	(	66	B	92	\	118	v
15	si	41	)	67	C	93	]	119	w
16	dle	42	*	68	D	94	^	120	x
17	dc1	43	+	69	E	95	_	121	y
18	dc2	44	,	70	F	96	`	122	z
19	dc3	45	–	71	G	97	a	123	{
20	dc4	46	.	72	H	98	b	124	|
21	nak	47	/	73	I	99	c	125	}
22	syn	48	0	74	J	100	d	126	~
23	etb	49	1	75	K	101	e	127	del
24	can	50	2	76	L	102	f		
25	em	51	3	77	M	103	g		
Коди 128-255									
128	Ђ	154	љ	180	ґ	206	О	232	и
129	Ѓ	155	>	181	µ	207	П	233	й
130	‚	156	њ	182	¶	208	Р	234	к
131	ѓ	157	ќ	183	·	209	С	235	л
132	"	158	ћ	184	ё	210	Т	236	м
133	…	159	џ	185	№	211	У	237	н
134	†	160		186	є	212	Ф	238	о
135	‡	161	Ў	187	»	213	Х	239	п
136	^	162	ў	188	j	214	Ц	240	р
137	‰	163	J	189	S	215	Ч	241	с
138	Љ	164	¤	190	s	216	Ш	242	т
139	<	165	Ґ	191	ї	217	Щ	243	у
140	Њ	166	¦	192	А	218	Ъ	244	ф
141	Ќ	167	§	193	Б	219	Ы	245	х
142	Ћ	168	Ё	194	В	220	Ь	246	ц

Код	Симв.	Код	Симв.	Код	Симв.	Код	Симв.	Код	Симв.
143	Ц	169	©	195	Г	221	Э	247	ч
144	ђ	170	Є	196	Д	222	Ю	248	ш
145	‘	171	«	197	Е	223	Я	249	щ
146	’	172	¬	198	Ж	224	а	250	ъ
147	“	173		199	З	225	б	251	ы
148	”	174	®	200	И	226	в	252	ь
149	•	175	Ї	201	Й	227	г	253	э
150	–	176	°	202	К	228	д	254	ю
151	—	177	±	203	Л	229	е	255	я
152	-	178	I	204	М	230	ж		
153	™	179	i	205	Н	231	з		

Кодова таблиця системи DOS

Код	Симв.	Код	Симв.	Код	Симв.	Код	Симв.	Код	Симв.
Коди 0-127									
0	nul	26	eof	52	4	78	N	104	h
1	soh	27	esc	53	5	79	O	105	i
2	stx	28	fs	54	6	80	P	106	j
3	etx	29	qs	55	7	81	Q	107	k
4	eot	30	rs	56	8	82	R	108	l
5	enq	31	us	57	9	83	S	109	m
6	ack	32	sp	58	:	84	T	110	n
7	bel	33	!	59	;	85	U	111	o
8	bs	34	"	60	<	86	V	112	p
9	tab	35	#	61	=	87	W	113	q
10	lf	36	$	62	>	88	X	114	r
11	vt	37	%	63	?	89	Y	115	s
12	np	38	&	64	@	90	Z	116	t
13	cr	39	'	65	A	91	[	117	u
14	so	40	(	66	B	92	\	118	v
15	si	41	)	67	C	93	]	119	w
16	dle	42	*	68	D	94	^	120	x
17	dc1	43	+	69	E	95	_	121	y
18	dc2	44	,	70	F	96	`	122	z
19	dc3	45	–	71	G	97	a	123	{
20	dc4	46	.	72	H	98	b	124	|
21	nak	47	/	73	I	99	c	125	}
22	syn	48	0	74	J	100	d	126	~
23	etb	49	1	75	K	101	e	127	del
24	can	50	2	76	L	102	f		
25	em	51	3	77	M	103	g		
Коди 128-255									
128	А	154	Ъ	180	┤	206	╬	232	ш
129	Б	155	Ы	181	╡	207	═	233	щ
130	В	156	Ь	182	╢	208	╨	234	ъ
131	Г	157	Э	183	╖	209	╤	235	ы
132	Д	158	Ю	184	╕	210	╥	236	ь
133	Е	159	Я	185	╣	211	╙	237	э

Код	Симв.	Код	Симв.	Код	Симв.	Код	Симв.	Код	Симв.
134	Ж	160	а	186	‖	212	╘	238	ю
135	З	161	б	187	╗	213	╒	239	я
136	И	162	в	188	╝	214	╓	240	Ё
137	Й	163	г	189	╜	215	╫	241	ё
138	К	164	д	190	╛	216	╪	242	Є
139	Л	165	е	191	┐	217	┘	243	є
140	М	166	ж	192	└	218	┌	244	Ї
141	Н	167	з	193	┴	219		245	ї
142	О	168	и	194	┬	220		246	Ў
143	П	169	й	195	├	221		247	ў
144	Р	170	к	196	─	222		248	°
145	С	171	л	197	┼	223		249	•
146	Т	172	м	198	╞	224	р	250	·
147	У	173	н	199	╟	225	с	251	√
148	Ф	174	о	200	╚	226	т	252	№
149	Х	175	п	201	╔	227	у	253	¤
150	Ц	176		202	╩	228	ф	254	■
151	Ч	177		203	╦	229	х	255	
152	Ш	178		204	╠	230	ц		
153	Щ	179		205	═	231	ч		

Додаток 2. Групування імен функцій по заголовним файлам (хайдерам)

1. <ctype.h> (<cctype>) – функції класифікації та перетворення символів

isalnum	Перевіряє, чи є символ буквою або цифрою
isalpha	Перевіряє, чи є символ буквою
iscntrl	Перевіряє, чи є символ керуючим
isdigit	Перевіряє, чи є символ цифрою
isgraph	Перевіряє, чи є символ видимим
islower	Перевіряє, чи є символ буквою нижнього регістру
isprint	Перевіряє, чи є символ друкованим
ispunct	Перевіряє, чи є символ символом пунктуації
isspace	Перевіряє, чи є символ розмежувальним
isupper	Перевіряє, чи є символ буквою верхнього регістру
tolower	Повертає символ в нижньому регістрі
toupper	Повертає символ у верхньому регістрі

2. <math.h> (<cmath>) – математичні функції

acos	Повертає арккосинус аргументу
asin	Повертає арксинус аргументу
atan	Повертає арктангенс аргументу
atan2	Повертає арктангенс відношення аргументів
ceil	Повертає найближче більше ціле число
cos	Обчислює косинус аргументу
cosh	Обчислює гіперболічний косинус аргументу
exp	Повертає значення e^x, де x – аргумент функції
fabs	Повертає модуль числа
floor	Повертає найближче менше ціле число.
fmod	Повертає залишок від ділення y (2-го аргументу) на x (1-й аргумент)
frexp	Виділяє з числа мантису та експоненціальну частину
ldexp	Перетворює мантису та показник ступеню в число
log	Обчислює натуральний логарифм
log10	Обчислює десятковий логарифм
modf	Розбиває число на цілу та дробову частини
pow	Підносить число до ступеня
sin	Обчислює синус аргументу
sinh	Обчислює гіперболічний синус аргументу
sqrt	Обчислює квадратний корінь аргументу
tan	Обчислює тангенс аргументу
tanh	Обчислює гіперболічний тангенс аргументу

3. <stdio.h> (<cstdio>) – функції введення/виведення

clearerr	Очищає прапори помилок при роботі з потоком
fclose	Закриває потік введення/виведення
feof	Перевіряє досягнення кінця файлу
ferror	Повертає код помилки при роботі з потоком
fflush	Записує дані з буфера
fgetc	Читає з потоку символ
fgetpos	Повертає поточну позицію у файлі
fgets	Читає з потоку рядок символів
fopen	Відкриває потік введення/виведення
fprintf	Записує дані в потік
fputc	Записує символ в потік
fputs	Записує рядок символів в потік
fread	Читає дані з потоку введення
freopen	Перевідкриває потік введення/виведення.
fscanf	Вводить з файлу форматовані дані
fseek	Переміщує позицію у файлі
fsetpos	Переміщує поточну позицію у файлі відносно його початку
ftell	Повертає поточну позицію у файлі
printf	Виводить рядок параметрів в певному форматі
putc	Записує символ в потік
putchar	Виводить символ на стандартний пристрій виведення
puts	Виводить рядок на стандартний пристрій виведення
remove	Видаляє файл з диску
rename	Перейменовує файл
rewind	Очищає прапори помилок при роботі з потоком та переходить до початку файлу
scanf	Вводить рядок параметрів в певному форматі
setbuf	Встановлює буферизацію потоку введення/виведення
sprintf	Виводить рядок параметрів в певному форматі
sscanf	Вводить дані з рядку
tmpfile	Відкриває потік бінарного введення/виведення в тимчасовий файл
tmpnam	Створює унікальне ім’я файлу
ungetc	Повертає символ в потік

4. <stdlib.h> (<cstdlib>)

abort	Перериває виконання програми
atof	Перетворює рядок в дійсне число
atoi	Перетворює рядок в ціле число (int)
atol	Перетворює рядок в довге ціле число (long)
exit	Перериває виконання програми
rand	Генерує випадкові числа від 0 до 2147483647
srand	Встановлює початкове псевдовипадкове число
system	Виконує рядок командного процесору ОС

5. <string.h> (<cstring>) – функції роботи з рядками

memchr	Шукає перше входження символу в блок пам'яті
memcmp	Порівнює блоки пам’яті
memcpy	Копіює блок пам’яті
memmove	Переносить блок пам’яті
memset	Заповнює блок пам’яті символом
strcat	Складає рядки
strchr	Шукає символ в рядку
strcmp	Порівнює рядки
strcpy	Копіює один рядок в інший
strcspn	Шукає входження одного з символів другого рядка в перший рядок
strlen	Повертає довжину рядка
strncat	Складає перший рядок з n символами другого
strncmp	Порівнює перший рядок з n символами другого
strncpy	Копіює перші n символів другого рядка в перший
strpbrk	Шукає один з символів другого рядка в першому
strrchr	Шукає останнє входження символу в рядку
strspn	Шукає символ першого рядка, який не входить у другий рядок
strstr	Шукає підрядок в рядку
strtok	Виділяє фрагменти рядка першого рядка, розділені одно- або багатосимвольними роздільниками з другого рядка.

6. <time.h> (<ctime>) – функції для роботи з датою та часом

asctime	Перетворює дату та час в рядок
clock	Повертає час виконання програми
ctime	Перетворює час у рядок
difftime	Повертає різницю часів
gmtime	Ініціалізує структуру tm на основі time_t
localtime	Ініціалізує структуру tm на основі time_t
mktime	Заповнює поля дня тижня і дня року
time	Повертає поточні дату та час у вигляді time_t (кількість секунд, починаючи з 1970 року)