МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій та комп’ютерної інженерії

Каштан В.Ю., Соколова Н.О.

Обчислювальна техніка та програмування (за професійним спрямуванням) для студентів спеціальності 141 Електроенергетика, електротехніка та електромеханіка

Практикум (1 частина)

Дніпро
НТУ “ДП”
2023

Каштан В.Ю., Соколова Н.О.

Обчислювальна техніка та програмування (за професійним спрямуванням) для студентів спеціальності 141 “Електроенергетика, електротехніка та електромеханіка”: практикум / Н.О. Соколова, В.Ю. Каштан. – Д.: НТУ «ДП», 2023. – 80 с.

Автори:

В.Ю. Каштан, к.т.н., доц., доцент кафедри інформаційних технологій та комп’ютерної інженерії.

Н.О. Соколова, к.т.н., доцент кафедри інформаційних технологій та комп’ютерної інженерії.

Затверджено до видання редакційною радою (протокол №6 від 27.06.2023 року за поданням методичної комісії спеціальності 141 Електроенергетика, електротехніка та електромеханіка (протокол №22/23-5 від 10.02.2023 року)

Практикум містить методичні вказівки, варіанти задач за професійним спрямуванням спеціальності 141 “Електроенергетика, електротехніка та електромеханіка” та призначений для самостійної роботи з кодом програми мовою С++ за професійним спрямуванням. Мета практикума – розвиток уміння вирішувати різноманітні завдання на ЕОМ.

Орієнтовано на активізацію навчальної діяльності бакалаврів для самостійної роботи та закріплення практичних знань з даної дисципліни.

Відповідальний за випуск завідувач кафедри інформаційних технологій та комп’ютерної інженерії В.В. Гнатушенко, д-р техн. наук, проф

ЗМІСТ

ВСТУП

Даний практикум надає допомогу студентам першого курсу, які вивчають дисципліну «Обчислювальна техніка та програмування» спеціальності 141 “Електроенергетика, електротехніка та електромеханіка”.

Для кращого розуміння можливостей різних операторів, а також вміння використовувати їх при розв’яанні практичних завдань запропоновано закріпити ці вмінні при розв’язанні задач за професійним спрямуванням спеціальності 141.

Пракимкум допомагає освоїти основні типи алгоритмів: лінійні, розгалужувані та циклічні; а також запис складних арифметичних та логічних виразів мовою С++.Навчитися обробляти однотипні дані, що задаються одновимірними або двовимірними масивами. Особлива увага приділяється типовим завданням при обробці масивів: обчислення загальних характеристик (суми, добутку та кількості елементів), пошук максимального або мінімального елемента, пошук заданого елементу, перестановка елементів, сортування.

Частина запропонованих завдань містить приклади рішення завдань. Їх слід виконувати в першу чергу. Потрібно набрати текст програми, скомпілювати, запустити на виконання, протестувати її роботу на різних наборах вхідних даних і порівняти отриманні результати.

Запропоновано виконувати завдання у порядку, в якому традиційно вивчаються відповідні розділи у курсі програмування. Перш ніж приступити до вирішення завдань, потрібно уважно причитати теоритечні відомості до даної теми. Якщо відразу впоратися із завданням не виходить, то можна переглянути рішення і потім ще раз спробувати вирішити завдання самостійно. Перед тим, як почати працювати на комп'ютері (набирати програму в редакторі коду), рекомендується розробити блок-схему алгоритму рішення задвдання на папері.

Завдання вважається вирішеним, якщо написана програма працює так, як сказано в умові задачі.

1 Знайомство з С++. Виконання програми простої структури

Мета: Знайомство з середовищем програмування, заголовними файлами, типами даних; створення, налагодження та виконання простих програм, що містять введення/виведення інформації і найпростіші обчислення.

Короткі теоретичні відомості

Мова С створена в 1972р. Деннісом Рітчі та Брайном Керніганом при розробці ОС Unix. Вона проектувалася як інструмент системного програмування з орієнтацією на розробку добре структурованих програм. Таким чином, він поєднує в собі, з одного боку, засоби мови програмування високого рівня: опис типів даних, оператори `for`, `while`, `if` і т.п., а, з іншого боку, містить засоби мови типу Асемблер: реєстрові змінні, адресну арифметику, можливість роботи з бітовими полями і т.д.

На початку 80-х років XX століття співробітником AT&T Bell Laboratories Бьорном Страуструпом розроблена мова С++, на основі процедурно-орієнтованої мови С. При створенні С ++ переслідувалися наступні цілі:

• підтримка абстрактних (тих, що визначаються користувачем) типів даних;

• надання засобів для об'єктно-орієнтованого програмування;

• поліпшення існуючих конструкцій мови С.

1.1. Структура програми

У загальному випадку програма на мові С++ має наступну структуру:

```
# директиви препроцесора
глобальні оператори опису
тип main () // основна функція
{ локальні оператори опису
  оператори
  return 0;
}
```

Вихідна програма, підготовлена на мові С ++ у вигляді текстового файлу проходить 3 етапи обробки:

1. препроцесорну перетворення тексту;
2. компіляція;
3. компоновка (редагування зв'язків, лінкування або зборка).

Після цих 3 етапів формується виконуваний машинний код програми.

Директиви препроцесора - управляють перетворенням тексту програми до її компіляції. Правила препроцесорної обробки визначає програміст за допомогою директив препроцесора. Директива починається з `#`. наприклад,

1) `#define` - вказує правила заміни в тексті.

```
#define ZERO 0.0
```

Це означає, що кожне використання в програмі імені `ZERO` буде замінюватися на 0.0.

2) `#include <ім'я заголовного файлу>` - призначена для включення в текст програми тексту з каталогу заголовних файлів, що поставляються разом зі стандартними бібліотеками. Кожна бібліотечна функція С ++ має відповідний опис в одному з заголовних файлів. Список заголовних файлів визначено стандартом мови. Вживання директиви `include` не підключає відповідну стандартну бібліотеку, а тільки дозволяє вставити в текст програми описи із зазначеного заголовного файлу. Підключення кодів бібліотеки здійснюється на етапі компонування, тобто після компіляції. Хоча в заголовних файлах містяться всі описи стандартних функцій, в код програми включаються тільки ті функції, які використовуються в програмі.

Після виконання препроцесорної обробки в тексті програми не залишається жодної препроцесорної директиви. Програма представляє собою набір описів і визначень, і складається з набору функцій. Серед цих функцій завжди повинна бути функція з ім'ям `main`. Без неї програма не може бути виконана. Перед ім'ям функції розміщаються відомості про тип значення, що повертається функцією (`int` або `void`).

```
int main()
{ оператори
  return 0;     // якщо програма нічого не повертає,
                // цей оператор можна опускати
}
```

Якщо функція нічого не повертає, то вказується тип `void`:

```
void main ( )
{ оператори }
```

Кожна функція, в тому числі і `main` повинна мати набір параметрів, він може бути порожнім, але дужки обов'язкові. Функція `main` може приймати тільки два параметра `int argc, char ** argv` (їх використання буде розбиратися далі).

За заголовком функції розміщується тіло функції. *Тіло функції* - це послідовність визначень, описів і виконуваних операторів, укладених у фігурні дужки. Кожне визначення, опис або оператор закінчується крапкою з комою.

Визначення вводять об'єкти (*об'єкт* - це іменована область пам'яті, окремий випадок об'єкта - *змінна*), необхідні для подання в програмі оброблюваних даних. прикладом є

```
const int y=10; //іменована константа
float x; //змінна
```

Описи повідомляють компілятор про властивості і імена об'єктів і функцій, описаних в інших частинах програми.

Оператори визначають дії програми на кожному кроці її виконання.

1.2 Перша програма

Незалежно від того, в якому середовищі програмування ви працюєте, для написання програми необхідно створити проект (в нашому випадку консольний додаток). Практично всі середовища програмування при створенні проекту пропонують шаблон програми.

Внесемо зміни в запропонований шаблон і наберемо наступний код:

```
// Лістинг 1.1.Первая програма
#include <iostream>
#include <cstdlib> // для паузи system
using namespace std; //декларування простору імен

int main()
{
    cout << "Hello, world!" << endl;
    system("pause"); //пауза
    return 0;
}
```

1.2.1 Опис синтаксиса

Директива `#include` використовується для підключення інших файлів в код. Рядок `#include <iostream>`, буде замінена вмістом файла «`iostream.h`», який знаходиться в стандартній бібліотеці мови і відповідає за введення і виведення даних на екран. `#include <cstdlib>` підключає стандартну бібліотеку мови С, яка містить в собі функції, які займаються виділенням пам'яті, контролем процесу виконання програми, перетворенням типів і інші. Потребує такого типу з'єднання для роботи функції `system`.

Вміст третього рядка - `using namespace std;` вказує на те, що ми використовуємо за замовчуванням стандартне простір імен з назвою «`std`». Все те, що знаходиться всередині фігурних дужок функції `int main () {}` буде автоматично виконуватися після запуску програми.

Рядок `cout << "Hello, world!" << endl;` говорить програмі виводити повідомлення з текстом `"Hello, world"` на екран. Оператор `cout` призначений для виведення тексту на екран командного рядка. Після нього ставляться дві кутові лапки (<<) - вказівка на виведення в потік. Далі йде текст, який буде виводитися. Він міститься в подвійних лапках. Оператор `endl` переводить рядок на рівень нижче (можна використовувати `"\ n"`).

Якщо в процесі виконання станеться якийсь збій, то виникне код помилки, відмінний від нуля. Якщо ж робота програми завершилася без збоїв, то код помилки дорівнюватиме нулю. Команда `return 0;` необхідна для того, щоб передати операційній системі повідомлення про вдале завершення програми.

У подальших прикладах підключення вже описаних заголовних модулів, опис простору імен і оператори `system ("pause");` і `return 0;` будемо опускати для стислості.

На рис. 1.1 зображено структуру організації файлів у типовому проекті мовою С та показано етапи створення модуля, що виконується, за участю цих файлів.

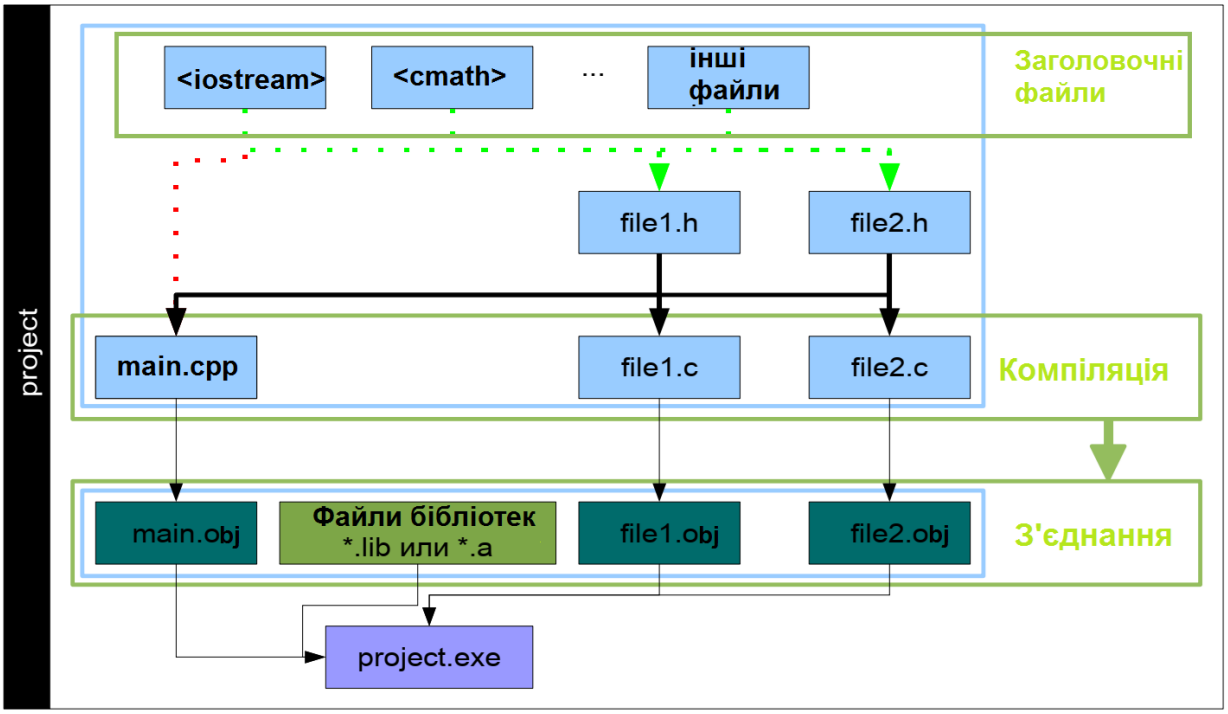

Рисунок 1.1 – Структура файлів С

Розглянемо приклад:

```
#include "add.h"

int main()
{
    cout << "The sum of 3 and 4 is " << add(3, 4) << '\n';
    return 0;
}
```

Коли препроцесор обробляє рядок `#include "add.h"`, він копіює вміст `add.h` в поточний файл у цю точку. Так як `add.h` містить попереднє оголошення для функції `add`, це попереднє оголошення буде скопійовано в `main.cpp`. Кінцевим результатом є програма, яка виводить значення The sum of 3 and 4 is. На рисунку 1.2 наведено графічне представлення процесу компіляції програми.

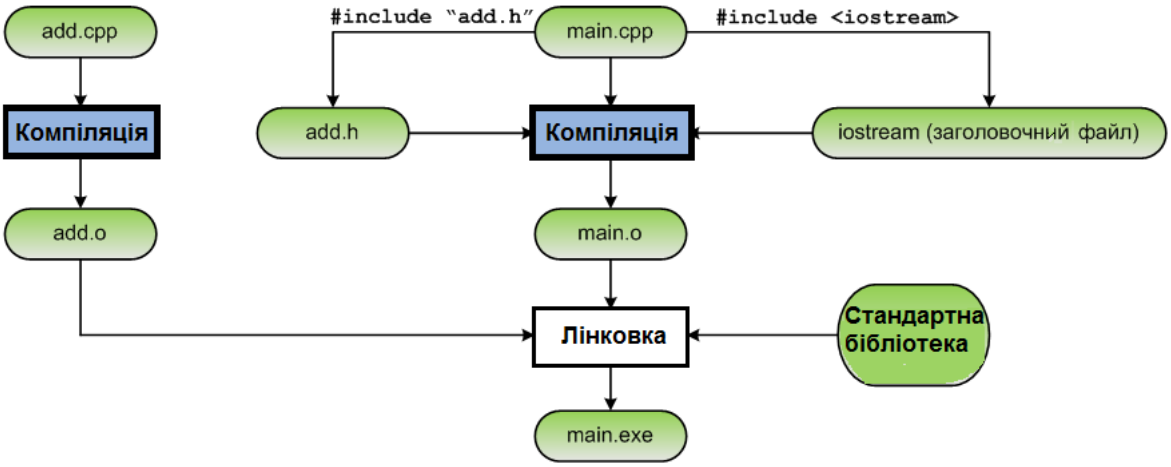

Рисунок 1.2 – Діаграма компіляції програми

1.2.2 Завдання. Компіляція та запуск

Скомпілюйте і запустіть програму із лістингу 1.1 (використовуючи пункти меню і відповідні їм піктографічні кнопки обраного середовища програмування).

1.3. Введення та виведення

1.3.1. Введення та виведення в стандартному С

Обмін даними із зовнішнім світом програма на стандартному С реалізує за допомогою бібліотеки функцій вводу-виводу

```
#include <stdio.h>
printf ( <форматний_рядок>,<список_аргументів>);//виведення
scanf ( <форматний_рядок>,<список_аргументів>); //введення
```

<форматний рядок> - рядок символів, укладених в лапки, який показує, як повинні бути надруковані аргументи. В якості аргументів використовуються змінні. Форматний рядок може містити:

- символи тексту;
- специфікації перетворення
- керуючі символи.

Кожному аргументу відповідає своя специфікація перетворення:

%d - десяткове ціле число;

%f - число з плаваючою точкою;

%c - символ;

%s - рядок;

\n - управляючий символ новий рядок.

наприклад:

```
printf ("Значення числа Пі дорівнює %f\n", pi);
scanf("%d%f", x, y);
```

1.3.2. Введення та виведення в C++

Використовується заголовний файл <iostream.h>, в якому визначені стандартні потоки введення даних з клавіатури cin і виведення даних на екран дисплея cout, а також відповідні операції: << - операція запису даних в потік; >> - операція читання даних з потоку.

Наприклад:

```
#include <iostream>;
cout << "\nВведіть кількість елементів: ";
cin >> n;
```

При виведенні кирилиці вона може відображатися некоректно. Хоча вже давно розроблений єдиний стандарт кодування символів - Unicode, в Windows досі використовуються кілька кодирувальних таблиць, а саме - cp866, cp1251. Використання декількох таблиць кодування символів і є причиною появи символів псевдографіки, замість кирилиці. Для коректного відображення кирилиці необхідно використовувати функцію setlocale (int type, const char * locale). Ця функція дозволяє запитувати або встановлювати певні параметри (локаль), які залежать від географічного положення. Ця функція визначена в заголовному файлі <clocale>, необхідність його підключення визначає середовище програмування.

Функція setlocale() має два параметри, перший параметр - тип категорії локалі, другий параметр - значення локалі. За замовчуванням, встановлена локаль: setlocale (LC_ALL, "C"). яка є нейтральною локаллю з мінімальною інформацією, що дозволяє передбачити результат програми. Вся локаль, яка визначається цією операційною системою, може бути встановлена викликом функції setlocale (LC_ALL, ""). Для виведення кирилиці можна використовувати такі виклики:

```
setlocale(LC_ALL, "Ukrainian");
setlocale(LC_ALL, "ukr");
setlocale(LC_ALL, "rus");
setlocale(LC_ALL, "Russian");
```

Якщо необхідно змінити частину поточної локалі, замість параметра LC_ALL визначається параметр певної категорії, так значення LC_TYPE встановлює тільки набір символів.

Функція setlocale() працює тільки з потоком виводу, для коректного введення кирилиці є кілька способів. Один з них це використання функцій SetConsoleCP (1251) і SetConsoleOutputCP (1251), для цього необхідно підключити до проекту заголовний файл <windows.h>. Аргументом цих функцій є ідентифікатор кодової сторінки, потрібна для кириліці сторінка win-cp 1251. Функція SetConsoleCP() встановлює потрібну кодову таблицю на потік введення, тоді як функція SetConsoleOutputCP() встановлює потрібну кодову таблицю на потік виведення.

1.4 Типи даних

Мова програмування С ++ є розширюваною мовою програмування. Поняття *розширювана* означає те, що крім вбудованих типів даних, можна створювати свої типи даних. Тому в С ++ існує величезна кількість типів даних.

Основні (фундаментальні, базові) типи в С ++ діляться на три категорії: цілочисельні, з плаваючою комою і `void`.

Типом `void` описується порожній набір значень. Завдання змінних типу `void` неможливо. Цей тип служить в основному для оголошення функцій, які не повертають значення, або універсальних вказівників на нетипізовані або довільно типізовані дані. Будь-який вираз можна явно перетворити або привести до типу `void`. Однак такі вирази можна використовувати тільки в наступних операторах і операндах:

- в операторі виразу;
- в лівому операнді оператора кома;
- в другому та третьому операндах тернарного оператора `(? :)`.

Сі ++ надає множину базових типів. Більшість з них формується за допомогою одного з чотирьох арифметичних специфікаторів типу - `char`, `int`, `float` і `double`, і опціональних специфікаторів - `signed`, `unsigned`, `short` і `long`. Основні типи даннихС ++ наведені в таблиці 1.

Таблиця 1.1 – Фундаментальні типи даних С++

Тип	Опис
	Цілі числа
`char`	`char` - це цілочисельний тип, зазвичай містить члени основної кодування виконання (за замовчуванням в Microsoft С ++ це кодування ASCII). Компілятор С ++ обробляє змінні типу `char`, `signed char` і `unsigned char` як змінні різних типів. Змінні типу `char` підвищуються до типу `int`, як якщо б за замовчуванням вони мали тип `signed char`, якщо не використовується параметр компіляції `/J`. У цьому випадку вони розглядаються як тип `unsigned char` і підвищуються до типу `int` без розширення знака.
`bool`	`bool` - це цілочисельний тип, який може мати одне з двох значень: `true` або `false`. Його розмір не визначений.
`short`	`short int` (або просто `short`) - це цілочисельний тип, розмір якого більше або дорівнює розміру типу `char` і менше або дорівнює розміру типу `int`. Об'єкти типу `short` можуть оголошуватися як об'єкти типу `signed short` або `unsigned short`. `signed short` - синонім `short`.
`int`	`int` - це цілочисельний тип, розмір якого більше або дорівнює розміру типу `short int` і менше або дорівнює розміру типу `long`. Об'єкти типу `int` можуть оголошуватися як об'єкти типу `signed int` або `unsigned int`. `signed int` - синонім `int`.
`long`	`long` (або `long int`) - це цілочисельний тип, розмір якого більше або дорівнює розміру типу `int`. Об'єкти типу `long` можуть оголошуватися як об'єкти типу `signed long` або `unsigned long`. `signed long` - синонім `long`.

Тип	Опис
long long	Більше, ніж unsigned long. «Наймолодший» з базових типів. Об'єкти типу long long можуть оголошуватися як об'єкти типу signed long long або unsigned long long. signed long long - синонім long long.
wchar_t	Змінна типу wchar_t позначає розширений символьний або мультибайтний символьний тип.
	З плаваючою комою
float	float — це тип з плаваючою комою найменшого розміру.
double	double — це тип з плаваючою комою, розмір якого більше або дорівнює розміру типу float, але менше або дорівнює розміру типу long double.
long double	long double — це тип з плаваючою комою, розмір якого більше або дорівнює розміру типу double. Для систем Майкрософт: подання long double і double ідентично. Однак типи long double і double - це окремі типи.

У стандарті С ++ розмір типів даних жорстко не визначено, дізнатися скільки відводить середовище розробки під кожен тип даних можна за допомогою операції sizeof (тип) . Стандарт С ++ гарантує, що:

```
1 == sizeof(char) <= sizeof(short) <= sizeof(int) <=
        <= sizeof(long) <= sizeof(long long)
```

Найбільш поширені розміри типів даних і діапазон їх значень наведені в таблиці 2.

Таблиця 1.2 – Розмір і діапазон значень типів даних

Тип даних	Назва	Розмір, біт	Діапазон значень
bool	булевський, логічний	8	false або true
unsigned char	символьний, беззнаковий цілий довжиною не менше 8 біт	8	0 . . 255
signed char (char)	символьний, цілий довжиною не м енше 8 біт	8	-128 . . 127
unsigned short	беззнаковий цілий, коротке цілочисове число без знака	16	0 . . 65535
short int (short)	короткий цілий, короткий цілочисловий, короткий цілочисловий зі знаком	16	-32768 . . 32767
unsigned int	беззнаковий цілий	32	0 . . 4 294 967 295
signed int (int)	цілий, signed	32	-2 147 483 648 . . 2 147 483 647
unsigned long	беззнаковий довгий цілий, довге целочислове число без знака	32	0 . . 4294967295
long	довгий цілий, довге целочислове зі знаком	32	-214748348 . . 2147483647
long long	довге довге ціле	64	-9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807
unsigned long long	без знака довге довге	64	0 .. 18 446 744 073 709 551 615

Тип даних	Назва	Розмір, біт	Діапазон значень
`float`	дійсний одинарної точності	32	3.4E-38 . . 3.4E+38 (7 знаков)
`double`	дійсний подвійної точності	64	1.7E-308 . . 1.7E+308
`long double`	дійсний максимальної точності	Не менее 64	Мінімум: 1.7E-308 . . 1.7E+308

Різні розробники середовищ програмування можуть визначати додаткові типи даних. Так в Visual Studio визначені наступні нестандартні типи даних:

`__int8`, `__int16`, `__int32`, `__int64`, `__int128`

ціле число із зазначенням розміру `__intn`, де `n` — розмір у бітах цілочисельної змінної. Доступність типів залежить від архітектури.

1.5 Константи та змінні

Константа - це значення, яке не може бути змінено. В C ++ константи бувають неіменованими і іменованими. Тип *неіменованої константи* визначається за її виглядом. Синтаксис мови визначає наступні типи різних неіменованих констант:

- символи;
- дійсні числа;
- цілі числа;
- константи зліченого типа;
- рядки;
- нульовий вказівник (NULL).

Таблиця 1.3 – Формати констант, відповідні до кожного типу

Константа	Формат	Приклади
Ціла	*Десятковий:* послідовність десяткових цифр, що починається не з нуля, якщо це не число нуль	8, 0, 199226
	Восьмирічний: нуль, за яким йдуть восьмеричні цифри (0,1,2,3,4,5,6,7)	01, 020, 07155
	Шістнадцятковий: 0x або 0X, за яким йдуть шістнадцяткові цифри	0xA, 0x1B8, 0X00FF
Дійсна	*Десятковий*: `[цифри].[цифри]` (можуть бути опущені або ціла частина, або дробова, але не обидві разом).	5.7, .001, 35.
	Експонентний: `[цифри][.][цифри]{E|e}[+|-][цифри]` (можуть бути опущені або ціла частина, або дробова, але не обидві разом. Якщо вказані	0.2E6, .11e-3, 5E10

Константа	Формат	Приклади

Символьна	Один або два символи, укладених в апострофи	клавіатурні:
		`'A', 'ю', '*'`
		кодові:
		`'\n', '\a', '\t', '\?'`
		кодово числові:
		`'\0', '\012', '\x07'`

Рядкова	Последовність символів, укладених в апострофи	`"Тут був Vasia",` `"\tЗначення r=\0xF5\n"`

Змінні можна змінювати. При завданні значення змінної у відповідну їй комірка пам'яті поміщається код цього значення. Доступ до значення можливий через ім'я змінної, а доступ до комірки пам'яті - за її адресою. Кожна змінна перед використанням в програмі повинна бути визначена, тобто їй повинна бути виділена пам'ять. Розмір комірки пам'яті, що виділяється для змінної, і інтерпретація її вмісту можуть відрізнятися залежно від типу, зазначеного у визначенні змінної. Найпростіша форма визначення змінних:

```
тип ім’я [ініціалізатор];
тип ім’я =ініціалізатор;
```

Відповідно до синтаксису мови змінні автоматичної пам'яті після визначення за замовчуванням мають невизначені значення. Змінним можна присвоювати початкові значення, явно вказуючи їх у визначеннях. Цей прийом називається *ініціалізацією*.

```
float pi=3.14, cc=1.3456;
unsigned int year=1999;
```

Іменовані константи С ++ оголошуються як змінні з модифікатором `const`, який показує, що значення даної змінної міняти не можна. Іменовані константи повинні бути ініційовані при оголошенні. Щоб в програмі відрізняти змінні від констант, рекомендується імена констант писати великими літерами.

```
const int N=20;
```

Для змінних дуже велике значення має місце їх введення в програму. В С ++ змінні можна описувати безпосередньо перед використанням, в залежності від місця опису будуть залежати *область дії, область видимості і час життя змінної*. Змінні, описані до функції `main()`, називають *глобальними*. Область дії змінної - з моменту її опису і до кінця блоку, в якому вона була описана. Область видимості змінної найчастіше збігається з її областю дії, за винятком випадку опису у вкладеному блоці змінної з тим же ім'ям.

```
//Лістинг 1.2. Опис змінних
int n;                          //1
int main()
{    cout<<"n=";                //2
     cin>>n;                    //3
     cout<<"n="<<n<<endl;       //4
     int n;                     //5
     cout<<"n=";                //6
     cin>>n;                    //7
     cout<<"n="<<n<<endl;       //8
     cout<<"n="<<::n<<endl;     //9
}
```

У лістингу 1.2 описані дві змінні `n`: в 1 рядку і 5. Змінна з 1 рядка буде глобальною, область її дії - до кінця програми, а ось область видимості не повністю збігається з областю дії: починаючи з 5 рядка, в дію вступає нова змінна `n`. Однак можна отримати можливість звертатися до «першої» `n`, використовуючи операцію доступу до області видимості `::`, що і продемонстровано в 9 рядку.

1.6 Операції

Операції і вирази задають певну послідовність дій, але не є закінченими конструкціями мови. Операції С ++ поділяються на:

Унарні:

`&`	отримання адреси операнда
`*`	звернення за адресою (розіменування)
`+`	унарний плюс
`-`	унарний мінус, змінює знак арифметичного операнда
`~`	порозрядне інвертування внутрішнього двійкового коду (побітовое заперечення)
`!`	логічне заперечення (НЕ). В якості логічних значень використовується 0 - брехня і не 0 - істина, запереченням 0 буде 1, запереченням будь-якого ненульового числа буде 0.
`++`	збільшення на одиницю: *префіксна* операція - збільшує операнд **до** його використання, *постфіксна* операція збільшує операнд **після** його використання.
`--`	зменшення на одиницю: *префіксна* операція - зменшує операнд **до** його використання, *постфіксна* операція зменшує операнд **після** його використання.
`sizeof`	обчислення розміру (в байтах) для об'єкта типу операнда
`new`	виділення пам'яті
`delete`	звільнення пам'яті
`::`	доступ до області видимості
`(тип)значення` `тип (значення)`	перетворення типу
`.`	вибір елемента за іменем
`->`	вибір елемента за вказівником
`[]`	вибір елемента за індексом

`.*`	вибір елемента за іменем через вказівник
`->*`	вибір елемента за вказівником через вказівник
`()`	виклик функції
`typeid`	ідентифікація типу (необхідні відомості про тип)
`const_cast`	перетворення константної змінної в неконстантну
`dynamic_cast`	динамічне перетворення типів (має сенс для поліморфних класів)
`static_cast`	перетворення одного типу в інший з контролем неприпустимого перетворення (наприклад, значення у вказівник)
`reinterpret_cast`	операція приведення типів даних, не повинна бути використана для приведення ієрархії класів або перетворення константних змінних

Бінарні операції:

Адитивні:

`+`	бінарний плюс (додавання арифметичних операндів)
`-`	бінарний мінус (віднімання арифметичних операндів)

Мультиплікативні:

`*`	множення операндів арифметичного типа
`/`	ділення (для цілочисельних операндів цілочисельне ділення)
`%`	отримання остачі від ділення цілочисельних операндів

Операції зсуву (визначені тільки для цілочисельних операндів).

Формат виразу з операцією зсуву:

```
операнд_лівий операція_зсуву операнд_правий
```

`<<`	зсув вліво бітового подання значення лівого цілочисельного операнда на кількість розрядів, що дорівнює значенню правого операнда
`>>`	зсув вправо бітового подання значення лівого цілочисельного операнда на кількість розрядів, що дорівнює значенню правого операнда

Поразрядні операції:

`&`	поразрядна кон'юнкція (І) бітового подання значень цілочисельних операндів
`|`	поразрядна диз'юнкція (АБО) бітового подання значень цілочисельних операндів
`^`	поразрядне виключне АБО бітового подання значень цілочисельних операндів

Логічні бінарні операції:

`&&`	кон'юнкція (І) цілочисельних операндів або відношень, цілочисельний результат брехня (0) або істина(1)
`||`	диз'юнкція (АБО) цілочисельних операндів або відношень, цілочисельний результат брехня (0) або істина(1)

Операції порівняння:

`<`	менше ніж
`>`	більше ніж
`<=`	менше або дорівнює
`>=`	більше або дорівнює
`==`	дорівнює
`!=`	не дорівнює

Операції присвоєння:

`=`	просте присвоєння
`*=`	присвоити результат множення
`/=`	присвоити результат ділення
`%=`	присвоити результат остачі від ділення
`+=`	присвоити результат додавання
`-=`	присвоити результат віднімання
`<<=`	присвоити результат зсуву вліво

>>=	присвоити результат зсуву вправо
&=	присвоити результат порозрядної (побітової) кон'юнкції
^=	присвоити результат порозрядного (побітового) виключного АБО
|=	присвоити результат порозрядної (побітової) диз'юнкції

Послідовність виразів:

,	Послідовне виконання виразів

Тернарна операція:

? :	якщо вираз перед ? не дорівнює 0, виконується оператор до : якщо дорівнює 0, виконується оператор після :

Операнд являє собою елемент-учасник операції. Операндами можуть бути константи, змінні, виклик функцій і вирази. *Вираз* - це послідовність знаків операції, операндів, круглих дужок, яка задає обчислювальний процес отримання результату певного типу. Найпростішим виразом є константа, змінна або виклик функції.

Порядок обчислення виразу визначається розташуванням знаків операцій, круглих дужок і пріоритетами виконання операцій; вирази з найвищим пріоритетом обчислюються першими.

Пріоритети операцій С ++ наведені в таблиці 1.4.

Таблиця 1.4 – Пріоритети операцій С++

Пріоритет	Знаки операцій	Назва операцій	Порядок виконання
1	::	доступ до області видимости	зліва
2	. ->[] () ++ -- typeid dynamic_cast static_cast reinterpret_cast const_cast	вибір елемента за іменем вибір елемента за вказівником вибір елемента за індексом виклик функції або конструювання значення постфіксний інкремент постфіксний декремент ідентифікація типу перетворення з перевіркою при виконанні перетворення з перевіркою при компіляції перетворення без перевірки константне перетворення	зліва
3	sizeof ++ -- ~ ! + - & * new delete (имя_типа)	розмір операнда в байтах префіксний інкремент префіксний декремент інверсія (поразрядне НЕ) логічне НЕ унарний плюс унарний мінус адреса розіменування виділення пам'яті або створення звільнення пам'яті або знищення перетворення типу	зправа
4	.* ->*	вибір елемента за іменем через вказівник вибір елемента за вказівником через вказівник	зліва

5	`*`
	`/`
	`%`
		множення
		ділення
		остача від ділення цілих (ділення за модулем)
			зліва

6	`+` `-`	додавання віднімання	зліва
7	`<<` `>>`	зсув вліво зсув вправо	зліва

8	`<`
	`>`
	`<=`
	`>=`
		менше
		більше
		менше або дорівнює
		більше або дорівнює
			зліва

9	`==` `!=`	дорівнює не дорівнює	зліва
10	`&`	поразрядне І	зліва
11	`^`	поразрядне виключне АБО	зліва
12	`|`	поразрядне АБО	зліва
13	`&&`	логічне І	зліва
14	`||`	логічне АБО	зліва
15	`? :`	умовна (тернарна)	зправа

16	`=`
	`*=, /=, %=`
	`+=, -=`
	`<<=, >>=`
	`&=, ^=, |=`
		присвоєння (просте та складені)	зправа

17	`throw`	генерація виключень	зправа
18	`,`	послідовність виразів	зліва

Операція визначення розміру `sizeof` призначена для обчислення розміру об'єкта або типу в байтах, і має дві форми:

```
sizeof вираз
sizeof (тип)
```

Наприклад:

```
float x = 1;
sizeof (float); //4
sizeof x; //4
sizeof (x+1.0); //8
```

Останній результат пов'язаний із тим, що дійсні константи за замовчуванням мають тип `double`. Дужки необхідні для того, щоб вираз, що стоїть у них, обчислювався раніше операції приведення типу, що має більший пріоритет, ніж додавання.

Ділення (/) та залишок від ділення (%).

Операція ділення застосовна до операндів арифметичного типу. Якщо обидва операнди цілі, результат операції округляється до цілого числа, в іншому випадку тип результату визначається правилами перетворення.

```
cout <<  << 9 / 5  << endl;        //1 (int)
cout << 9.0 / 5.0 << endl;         //1.80000 (double)
cout << << 9.0 / 5  << endl;       //1.80000 (double)
cout << 1.e7 / 9.0 <<  endl;       //1111111.111111
cout << 1.e7f / 9.0f <<  endl;     //1111111.125000(float)
```

В результаті ділення цілого числа 9 на ціле число 5 було отримано ціле число 1. Дробна частина від ділення 4/5 (0.8) відкидається. Наступні два рядки показують, що якщо хоча б один з операндів має формат з плаваючою точкою, шуканий результат (1.8) також буде представлений в цьому форматі. При комбінуванні змішаних типів C++ перетворює їх до одного типу. Відносна точність у двох останніх рядках виводу показує, що результат має тип `double`, якщо обидва операнди мають тип `double`, і тип `float`, якщо обидва операнди мають тип `float`. Константи у форматі з плаваючою точкою мають тип `double` за замовчуванням.

Операція залишку від ділення застосовується лише до цілих операндів. Знак результату залежить від реалізації.

Операція відносини (<, <=, >, >=, ==, !=) порівнюють перший операнд з другим. Операнди може бути арифметичного типу чи покажчиками. Результатом операції є значення true або false (будь-яке значення не дорівнює нулю, інтерпретується як true). Операції порівняння рівність і нерівність мають менший пріоритет, ніж інші операції сравнения.

!!! Зверніть увагу на різницю між операцією перевірки на рівність (==) та операцією присвоєння (=), результатом якої є значення, надане лівому операнду.

Операція заперечення (-, ! та ~).

Арифметичне заперечення (унарний мінус -) змінює знак операнда цілого чи дійсного типу протилежний.

Логічне заперечення (!) дає в результаті значення 0 якщо операнд є істина (не нуль), і значення 1, якщо операнд дорівнює нулю. Операнд повинен бути цілого або дійсного типу, а може мати тип вказівник (покажчик).

Порозрядне заперечення (~), часто зване побітовим, інвертує кожен розряд у двійковому поданні цілісного операнда.

Логічні операції

Порозрядні операції (&, |, ^) *застосовуються тільки до цілих операндів* і працюють з їх двійковим поданням. При виконанні операцій операнди зіставляються побітово (перший біт першого операнда з першим бітом другого, другий біт першого операнда з другим бітом другого і т.д.).

При *порозрядній кон'юнкції*, або порозрядному І (операція позначається &) біт результату дорівнює 1 тільки тоді, коли відповідні біти обох операндів рівні 1.

При *порозрядній диз'юнкції*, або порозрядному АБО (операція позначається |) біт результату дорівнює 1 тоді, коли відповідний біт хоча б одного з опе- рандів дорівнює 1.

При *порозрядному виключаючому АБО* (операція позначається ^) біт результату дорівнює 1 тільки тоді, коли відповідний біт тільки одного з операндів дорівнює 1.

```
6 & 5=4                    5 & 3=1
6 | 5=7                    5 | 3=7
6 ^ 5=3                    5 ^ 3=6
```

Логічні операції (&& та ||). Операнди логічних операцій І (&&) та АБО (||) можуть мати арифметичний тип або бути вкзівниками (покажчиками), при цьому операнди можуть бути різних типів. Перетворення типів немає, кожен операнд оцінюється з погляду його еквівалентності нулю (операнд рівний нулю, сприймається як `false`, не рівний нулю – `true`).

Результатом логічної операції є `true` чи `false`. Логічні операції виконуються зліва направо. Якщо значення першого операнда достатньо для визначення результату операції, другий операнд не обчислюється.

Операції зсуву (<<i>>) застосовуються до цілих операндів. Вони зсувають двійкове уявлення першого операнда вліво або вправо на кількість двійкових розрядів, задане другим операндом. При зсуві вліво (<<) розряди, що звільнилися, обнулюються. При зрушенні вправо (>>) біти, що звільнилися, заповнюються нулями, якщо перший операнд беззнакового типу, і знаковим розрядом в іншому випадку. Операції зсуву не враховують переповнення та втрату значущості. По суті, кожен зсув вліво аналогічний множенню на 2, зсув вправо – діленню на 2 (операції зсуву швидші ніж множення та ділення).

```
5>>=1
5<<2=20
-6>>5=-1
1<<7=-128
```

Дивовижний на першій погляд останній результат. Пояснюється це наступною обставиною. Після зсуву вліво на 7 позицій з числа 00000001 отримуємо число 1000000. Це негативне число, про що свідчить старший одиничний біт. Переводячи це число в десяткову систему, спочатку інвертуємо бінарний код і отримуємо 01111111. Цей код числа 127. Щоб отримати кінцеве значення, необхідно додати до цього результату 1 і додати мінус - в результаті приходимо до значення -128.

Операція (оператор) Кома

Операція Кома (або «оператор Comma») дозволяє обробляти кілька виразів (у той час, коли зазвичай дозволяється тільки одне).

```
x, y
```

Вираз, у якому знаходиться цей оператор, матиме значення правого операнда. Наприклад:

```
int x = 0;
int y = 2;
int z = (++x, ++y); // інкремент змінних x та y
```

Змінній `z` буде надано результат обчислення `++y` (правого операнда), що дорівнює 3.

Майже в кожному випадку, стейтмент, в якому є оператор Кома, краще записувати у вигляді окремих інструкцій. Наведений вище код коректніше буде записати наступним чином:

```
int x = 0;
int y = 2;
++x;
++y;
int z = y;
```

Зверніть увагу, оператор Кома має найнижчий пріоритет з усіх операторів (навіть нижче, ніж у оператора присвоєння), тому наступні два рядки коду роблять не одне й те саме:

```
z = (a, b);
z = a, b;
```

Спочатку обчислюється вираз `(a, b)`, який дорівнює значенню `b`, а потім результат присвоюється змінній `z` обчислюється як `(z = a), b`, тому змінній `z` надається значення `a`, змінна `b` - ігнорується

Більшість програмістів не використовують оператора Comma взагалі (хіба що тільки в циклах for).

Правило: *Уникайте використання оператора Comma (за винятком циклів for)*.

!!!!!Зверніть увагу, кома, яка використовується у викликах функцій, не є оператором Comma:

```
int sum = add(x, y); // Ця кома не є оператором Comma
```

Аналогічно, при оголошенні кількох змінних в одному рядку, кома використовується як роздільник, а не як оператор:

```
int x(3), y(5); // Ця кома не є оператором Comma
```

1.7 Вирази

З констант, змінних, роздільників і знаків операцій можна конструювати вирази. Кожен вираз складається з одного або декількох операндів, символів операцій і обмежувачів, в якості яких найчастіше виступають круглі дужки. Якщо вираз формує як результат ціле або дійсне число, то це *арифметичний вираз*. У С ++ визначені арифметичні операції:

+ - додавання;

- - віднімання;

* - множення;

/ - ділення;

% - остача від ділення.

Відношення - це пара арифметичних виразів, об'єднаних знаком операції відносин. В С ++ прийнято, що відношення має ненульове значення, якщо воно істинне і 0, якщо воно хибне.

У виразах важливий тип отримуваного значення. В С ++ тип результата залежить в першу чергу від типу операндів. Якщо додавання, віднімання і множення цілих чисел не призводить до жодних несподіванок, то при виконанні операції ділення можливі цікаві моменти. Оператор

```
cout<<"5/2="<<5/2<<endl;
```

виведе на екран 2, так як обидва операнда цілі, то і результат буде цілим, по суті, ділення цілих чисел - це ділення націло з остачею. Щоб отримати результат точний, необхідно, щоб хоча б один операнд був дійсним:

```
cout<<"5/2="<<5/2.0<<endl;
```

Для отримання остачі від ділення одного цілого числа на інше використовується операція %:

```
cout<<"5%2="<<5%2;
```

Так як символьний і булівський тип є цілочисельними типами, вони можуть використовуватися в арифметичних виразах, тобто символи можна складати!

```
// Лістинг 1.3. Арифметичні дії над символами
     char c,b;
     cout<<"c=";
     cin>>c;
     cout<<"b=";
     cin>>b;
     cout<<"c="<<c<<endl;
     cout<<"b="<<b<<endl;
     cout<<"c+b="<<c+b<<"\n";
...
```

У лістингу 1.3 оголошені дві змінні символьного типу `c` і `b`. Після введення символів виконується операція додавання, і результат виводиться на екран. Найцікавіше, що ввівши з клавіатури цифри 5 і 6, в результаті ми не отримаємо 11!

Також можна складати і змінні булівського типу. Діапазон допустимих значень типу даних `bool` від 0 до 255, який зіставлення з певними в мові програмування логічними константами `true` і `false` наступним чином: константі `true` еквівалентні все числа від 1 до 255 включно, тоді як константі `false` еквівалентно тільки одне ціле число - 0. Розглянемо програму з використанням типу даних `bool`.

```
// Лістинг 1.4. Тип даних bool
bool boolean = 25; // змінна типа bool з іменем boolean
if ( boolean ) // умова оператора if
     cout << "true = "  << boolean << endl;
          // виконається якщо умова істинна
     else cout << "false = " << boolean << endl;
          // виконається якщо умова хибна
```

У лістингу 1.4 оголошена змінна `boolean` типу `bool`, яка ініціалізована значенням 25. Теоретично після цього в змінній `boolean` повинно міститися число 25, але насправді в цій змінній міститься число 1, тому що у змінній типу `bool` можуть міститися два значення - 0 (`брехня`) або 1 (`істина`). Тоді як під тип даних `bool` відводиться цілий байт, а це значить, що змінна типу `bool` може містити числа від 0 до 255. Для визначення хибного й істинного значень необхідно всього два значення 0 і 1. Домовилися використовувати

числа від 2 до 255 як еквівалент числа 1, тобто істина, тому змінній `boolean` можна присвоїти число 25. Далі оператор умовного вибору `if` виводить на екран значення змінної `boolean`.

Якщо у виразі міститься кілька операцій одного пріоритету на одному й тому рівні, їх обробка відбувається відповідно до порядку виконання: зправа наліво чи зліва направо.

```
float logs = 120/4 * 5; // Яким буде результат?
```

У тому випадку, коли дві операції мають однаковий рівень пріоритету, C++ аналізує їхню асоціативність зліва направо або зправа наліво. Асоціативність зліва направо означає, що дві операції, що виконуються над тим самим операндом, мають однаковий пріоритет, то спочатку виконується операція зліва від операнда. У разі асоціативності зправа наліво першою буде виконано операцію праворуч від операнда. Якщо в одному виразі записано кілька операцій однакового пріоритету, унарні операції, умовна операція та операції присвоєння виконуються праворуч наліво, решта – зліва направо. Наприклад, `a=b=c` означає `a=(b=c)`, а `a+b+c` означає `(a+b)+c`. Слід зазначити, що пріоритет виконання та правила асоціативності діють лише тоді, коли дві операції відносяться до одного й того ж операнда.

Розглянемо такий вираз:

```
int dos = 20*5 + 24*6;
```

Відповідно до пріоритету виконання операцій, програма повинна помножити 20 на 5, потім помножити 24 на 6, після чого виконати додавання. Однак ні рівень пріоритету виконання, ні асоціативність не допоможуть визначити, яка з операцій множення має бути виконана насамперед. Можна було б припустити, що відповідно до властивості асоціативності повинна бути виконана операція зліва, проте в цьому випадку дві операції множення не належать до одного і того ж операнда, тому ці правила тут не можуть бути застосовані. Насправді, вибір порядку виконання операцій, який буде прийнятним для системи, залишений за конкретною реалізацією C++. В даному випадку при будь-якому порядку виконання буде отримано той самий результат.

Порядок обчислення виразів усередині виразів не визначено: наприклад, не можна вважати, що у виразі `(sin(x+2)+cos(y+1))` звернення до синуса буде виконано раніше, ніж до косінусу, і що `x+2` буде обчислено раніше ніж `y+1`.

```
y=x/2+a*5%15 //y=(x/12)+((a*5)%15)
```

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

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

то вираз `a+b` має значення 7 і тип `int`, а вираз `a=b` має значення, що дорівнює поміщеному в змінну `a` (в даному випадку 5) і тип, що збігається з типом цієї змінної.

Особливе місце в C/C++ займають операції інкремента і декремента. Інкремент `++` - це збільшення на одиницю. Декремент `--` - це зменшення на одиницю. Операції декремента і инкремента з легкістю замінюються

арифметичними операціями або операціями присвоювання. Але використовувати операції інкремента і декремента набагато зручніше.

```
//синтаксис операцій інкремента и декремента
++змінна;  // префіксний інкремент (преінкремент)
змінна++; // постфіксний інкремент (постінкремент)
--змінна;  // префіксний декремент (предекремент)
змінна--; // постфіксний декремент (постдекремент)
```

Коли операція інкремента або декремента ставиться перед ім'ям змінної, то така операція називається *префіксним инкрементом* (скорочено - *преінкрементом*) або *префіксним декрементом* (скорочено - *предекрементом*). А якщо операція інкремента або декремента ставиться після імені змінної, то така операція називається операцією *постфіксного інкремента* (скорочено - *постінкрементом*) або *постфіксного декремента* (скорочено - *постдекремент*). При використанні операції преінкремента значення змінної, перш за все, збільшується на 1, а потім використовується в виразі (предекремент -1). При використанні операції постінкременті значення змінної спочатку використовується у виразі, а потім збільшується на 1 (предекремент -1). У таблиці 5 показані приклади виразів з використанням операцій інкремента і декремента, а також наведена їх коротка характеристика, а в лістингу 1.5 приклади використання операцій інкремента/декремента і операцій порівняння.

Таблиця 1.5 – Операції інкремента и декремента в С++

Операція	Знак	Приклад	Коротке пояснення
преінкремент	++	`cout<<++value;`	Значення в змінній `value` збільшується, після чого оператор `cout` друкує це значення
предекремент	—	`cout<<—value;`	Значення в змінній `value` зменшується, після чого оператор `cout` друкує це значення
постінкремент	++	`cout<<value++;`	Оператор `cout` друкує значення змінної `value`, потім збільшує це значення на 1
постдекремент	—	`cout<<value—;`	Оператор `cout` друкує значення змінної `value`, потім зменшує це значення на 1

```
// Листинг 1.5. Приклади операцій інкремента и декремента
int main()
{int m=3, n=4;
 cout<<"n="<<n<<" m="<<m<<endl;
 cout<<"n+++m="<<n+++m<<" n="<<n<<" m="<<m<<endl;
 cout<<"n+++n="<<n+++n<<" n="<<n<<" m="<<m<<endl;
 cout<<"(m-- >n)="<<(m-- >n)<<" n="<<n<<" m="<<m<<endl;
 cout<<"(n-->m)="<<(n-->m)<<" n="<<n<<" m="<<m<<endl;
 cout<<"(--m-++n)="<<(--m-++n)<<" n="<<n<<" m="<<m<<endl;
 cout<<"(m*n<n++)="<<(m*n<n++)<<" n="<<n<<" m="<<m<<endl;
 cout<<"(n-->m++)="<<(n-->m++)<<" n="<<n<<" m="<<m<<endl;
 cout<<"(m*n<n++)+(n-- > m++)="<<(m*n<n++)+(n-- > m++);
 cout<<" n="<<n<<" m="<<m<<endl;
 cout<<"(m*n>n++)+(n-- > m++)="<<(m*n>n++)+(n-- > m++);
 cout<<" n="<<n<<" m="<<m<<endl; }
```

Результат роботи програми

```
n=4 m=3
n+++m=7 n=5 m=3
n+++n=10 n=6 m=3
(m-- >n)=0 n=6 m=2
(n-->m)=1 n=5 m=2
(--m-++n)=-5 n=6 m=1
(m*n<n++)=0 n=7 m=1
(n-- > m++)=1 n=6 m=2
(m*n<n++)+(n-- > m++)=1 n=6 m=3
(m*n>n++)+(n-- > m++)=2 n=6 m=4
```

1.8 Присвоєння

Оператори (*операції*) присвоєння зберігають значення в об'єкті, позначеному лівим операндом. Існує два типи операторів присвоювання: просте присвоєння, при якому значення другого операнда зберігається в об'єкті, заданому першим операндом, і складене присвоєння, при якому спочатку виконується або арифметична, або побітова операція, або операція зсуву, а потім зберігається результат. Всі оператори присвоєння наведені в таблиці 6, за винятком оператора =, є складеними.

Таблиця 1.6 – Оператори присвоєння C++

ОПЕРАТОР	ЗНАЧЕННЯ
=	Збереження даних другого операнда в об'єкті, зазначеним першим операндом (просте присвоювання).
*=	Множення значення першого операнда на значення другого операнда; збереження результату в об'єкт, зазначений першим операндом.
/=	Ділення значення першого операнда на значення другого операнда; збереження результату в об'єкт, зазначений першим операндом.
%=	Ділення за модулем першого операнда на значення другого операнда; збереження результату в об'єкт, зазначений першим операндом.
+=	Додавання значення першого операнда зі значенням другого операнда; збереження результату в об'єкт, зазначений першим операндом.
−=	Віднімання значення другого операнда зі значення першого операнда; збереження результату в об'єкт, зазначений першим операндом.
<<=	Зсув значення першого операнда вліво на кількість бітів, заданих значенням другого операнда; збереження результату в об'єкт, зазначений першим операндом.
>>=	Зсув значення першого операнда вправо на кількість бітів, заданих значенням другого операнда; збереження результату в об'єкт, зазначений першим операндом.
&=	Виконання операції побітового І для значень першого і другого операндів; збереження результату в об'єкт, зазначений першим операндом.
^=	Виконання операції побітового виключного АБО для значень першого і другого операндів; збереження результату в об'єкт, зазначений першим операндом.

У складних операціях присвоєння (`+=`, `*=`, `/=` і т.д.) при обчисленні виразу, що стоїть у правій частині, використовується значення з лівої частини, Вираз `a+=b` є компактнішим записом виразу `a=a+b`. У C++ існує

досить зручний формат використання арифметичних операторів - так звана скорочена форма арифметичних операторів. Відповідно до скороченої форми операторів команди виду `операнд1=операнд1 оператор операнд2` можна записувати у вигляді `операнд1 оператор=операнд2`. У разі якщо `операнд1` і `операнд2` - операнди вирази, а `оператор` - один із бінарних арифметичних операторів. Наприклад, команда `x+=3` еквівалентна команді `x=x+3`, а команду `a=a*b` можна записати як `a*=b`. Скорочена форма арифметичних операторів дозволяє суттєво спростити код і часто використовується практично.

!!!При присвоєнні проводиться перетворення типу виразу до типу лівої частини, що може призвести до втрати інформації.

Головна особливість оператора присвоювання C++ полягає в тому, що він повертає значення. Це означає, що вираз з оператором присвоювання може, своєю чергою, бути частиною іншого виразу. Тому крім традиційних присвоювань типу `n=5` допускається вираз множинного присвоєння: `a=в=c=d=t`. В результаті обчислення даного виразу змінні `a, b, c, d` приймуть значення `t`, якщо всі вони одного типу. Інструкції можуть бути і більш хитромудрими:

`n=(m=6)+3` присвоюються значення 9 та 6 відповідно.

Ще одна особливість оператора присвоєння пов'язана з перетворенням типу у виразах, що містять цей оператор. Якщо в інструкції виду `змінна = вираз` тип результату, що повертається виразом, не збігається з типом змінної, має місце автоматичне перетворення типів. Результат такої операції істотно залежить від того, з якого в який тип даних здійснюється перетворення. При цьому тип значення виразу приводиться у відповідність до типу змінної.

В C++ інтерес викликає множиннне присвоєння и перетвореня типів, яке при цьому відбувається.

```
//Лістинг 1.6. Множинне присвоєння
int main(int argc, char** argv)
{   int a=1,b=2,c=3;
    double y=5.7;
    b=c;                //1
    a=b;                //2
    cout<<"a="<<a<<" b="<<b<<" c="<<c<<endl;
    a=1; b=2; c=3;
    a=b=c;              //3
    cout<<"a="<<a<<" b="<<b<<" c="<<c<<endl;
    a=y=b=c=y;          //4
    cout<<"a="<<a<<" b="<<b<<" c="<<c<<" y="<<y;
}
```

Результат роботи програми:

```
a=3 b=3 c=3
a=3 b=3 c=3
a=5 b=5 c=5 y=5
```

Як показують результати роботи програми, рядок 3 повертає такий же результат, як и рядки 1-2. Цікавий результат дає рядок 4, в результаті виконання якого дійсна змінна `y` набуває цілого значення! Це відбувається тому, що в результаті присвоєння целій змінній дійсного значення відбувається втрата дробової частини.

1.9 Заголовний файл cmath (math.h)

Бібліотека `cmath` визначає набір функцій для виконання загальних математичних операцій и перетворень (таблица 7).

Таблиця 1.7 – Математичні функції С++

Тригонометричні функції

cos(x)	Обчислення косинуса кута, вираженого у радіанах.
sin(x)	Обчислення синуса кута, вираженого у радіанах.
tan(x)	Обчислення тангенса кута, вираженого у радіанах.
acos(x)	Обчислення арккосинуса, результат буде у радіанах.
asin(x)	Обчислення арксинуса, результат буде у радіанах.
atan(x)	Обчислення арктангенса, результат буде у радіанах.
atan2(x,y)	Обчислення арктангенса та квадранта за координатами x и y, результат буде у радіанах.

Гіперболічні функції

cosh(x)	Обчислення гіперболічного косинуса.
sinh(x)	Обчислення гіперболічного синуса.
tanh(x)	Обчислення гіперболічного тангенса.

Експонентні та логарифмічні функції

exp (x)	Обчислення експоненти.
frexp(x, exp)	Отримати мантису та показник степеня двійки.
ldexp(x,exp)	Генерація числа за значенням мантиси та показника степеня.
log (x)	Натуральний логарифм.
log10 (x)	Десятковий логарифм.
modf(x)	Розподіл дійсного значення на дробову та цілу частини.

Функції степеня

pow (x,n)	Піднесення числа до степеня.
sqrt (x)	Корінь квадратний.
cbrt(x)	Корінь кубічний (С++11)

Округлення, модуль та інші функції

ceil(x)	Округлення до найбільшого цілого значення.
abs (x)	Обчислити модуль значення.
floor(x)	Округлення до найменшего цілого значення.
fmod(a,b)	Остача від ділення a на b.
fabs (x)	Обчислити модуль значення.

Функції мінімума, максимума та різниці

fdim(x,y)	Додатня різниця – С++11
fmax(x,y)	Максимальне значення – С++11
fmin(x,y)	Мінімальне значення – С++11

Аргументи функцій дійсні. Стандарт С ++ 11 розширив значно перелік функцій `cmath`. Приклади використання функцій `cmath` приведений в лістингу 1.7.

```
// Лістинг 1.7. Використання математичних функцій
#include <cmath>
int main()
{  cout << "log10(10)     = " << log10(10.0)   << endl;
     // логарифм десятковий  10
   cout << "log10(1)      = " << log10(1.0)    << endl;
     // логарифм десятковий  1
   cout << "log(2.718281) = " << log(2.718281) << endl;
     // натуральний логарифм
   cout << "sqrt(9)       = " << sqrt(9.0)     << endl;
     // корінь квадратний
   cout << "pow(2,3)      = " << pow(2.0,3.0)  << endl;
     // два в кубі
   cout << "abs(0)        = " << abs(0.0)      << endl;
     // модуль нуля
   cout << "abs(-5)       = " << abs(-5.0)     << endl;
     // модуль -5
   cout << "ceil(3.14)    = " << ceil(3.14)    << endl;
     //округлення 3.14 до найбільшого цілого
   cout << "ceil(-2.4)    = " << ceil(-2.4)    << endl;
     // округлення -2.4 до найбільшого цілого
   cout << "floor(3.14)   = " << floor(3.14)   << endl;
     // округлення 3.14 до найменшого целого
   cout << "floor(-2.4)   = " << floor(-2.4)   << endl;
     // округлення -2.4 до найменшого целого
   cout << "fmod(2.4/2.0)   = " << fmod(2.4,2.0) << endl;
     // остача від ділення 2.4/2
}
```

Результат роботи програми:

```
log10(10)     = 1
log10(1)      = 0
log(2.718281) = 1
sqrt(9)       = 3
pow(2,3)      = 8
abs(0)        = 0
abs(-5)       = 5
ceil(3.14)    = 4
ceil(-2.4)    = -2
floor(3.14)   = 3
floor(-2.4)   = -3
fmod(2.4/2.0)   = 0.4
```

Розглянемо тепер, як формуються діапазони прийнятих значень і розміри займаної пам'яті. Максимальне значення деякого типу даних обчислюється за формулою:
для знакових типів даних (1):

$$max\_val\_type = 2^{b*8-1}-1; \qquad (1)$$

для беззнакових типів даних:

$$max\_val\_type = 2^{b*8}-1; \quad (2)$$

де, b - кількість байт, що виділяється в пам'яті під змінну з таким типом даних. Множимо на 8 (в одному байті 8 біт), для знакових типів віднімаємо 1 в показнику ступеня, так як старший біт відводиться на зберігання знака, і відповідно зменшується діапазон значень і ділиться на позитивні і негативні значення; віднімаємо з отриманого числа 1, так як діапазон чисел починається з нуля.

```
// Лістинг 1.8.Размер і максимальні значення базових типів
// даних
     // бібліотека маніпулювання введенням / виведенням
#include <iomanip>
     // заголовний файл математичних функцій
#include <cmath>
int main(int argc, char* argv[])
{    /*обчислюємо максимальне значення для типа даних bool*/
 cout<<"bool="<<sizeof(bool)<<""<<fixed<<setprecision(2)<<endl;
 cout<<"max value="<<(pow(2,sizeof(bool)*8.0)-1)<< endl;
     /*обчислюємо максимальне значення для типа даних char*/
 cout<<"char="<<sizeof(char)<<" "<<fixed<<setprecision(2);
 cout<<endl;
 cout<<"max value="<<(pow(2,sizeof(char)*8.0)-1)<<endl;
     /*обчислюємо максимальне значення для типа даних short int*/
 cout<<"short int="<<sizeof(short int);
 cout<<" "<<fixed<<setprecision(2)<<endl;
 cout<<"max value="<<(pow(2,sizeof(short int)*8.0-1)-1)<< endl;
     /*обчислюємо максимальне значення для типа даних unsigned
     short int*/
 cout<<"unsigned short int="<<sizeof(unsigned short int);
 cout<<" "<<fixed<<setprecision(2)<<endl;
 cout<<"max value="<<(pow(2,sizeof(unsigned short int)*8.0)-1);
 cout<<endl;
     /*обчислюємо максимальне значення для типа даних int*/
 cout<<"int="<<sizeof(int)<<" "<<fixed<<setprecision(2)<<endl;
 cout <<"max value="<<(pow(2,sizeof(int)*8.0-1)-1)<< endl;
     /*обчислюємо максимальне значення для типа даних unsigned
     int*/
 cout<<"unsigned int="<<sizeof(unsigned int);
 cout<<" "<<fixed<<setprecision(2)<<endl;
 cout<<"max value="<<(pow(2,sizeof(unsigned int)*8.0)-1)<<endl;
     /*обчислюємо максимальне значення для типа даних long int*/
 cout<<"long int="<<sizeof(long int);
 cout<<" "<<fixed<<setprecision(2)<<endl;
 cout<<"max value="<<(pow(2,sizeof(long int) * 8.0 - 1) - 1);
 cout<< endl;
     /*обчислюємо максимальне значення для типа даних undigned
     long int*/
 cout<<"unsigned long int="<<sizeof(unsigned long int);
 cout<<" "<<fixed<<setprecision(2)<<endl;
 cout<<"max value="<<(pow(2,sizeof(unsigned long int)*8.0)-1);
 cout<<endl;
```

```
    /*обчислюємо максимальне значення для типа даних float*/
 cout<<"float="<< sizeof(float)<<" "<<fixed<<setprecision(2);
 cout<<endl;
 cout<<"max value="<<(pow(2,sizeof(float)*8.0-1)-1)<<endl;
    /*обчислюємо максимальне значення для типа даних double*/
 cout<<"double="<<sizeof(double)<<" "<<fixed<<setprecision(2);
 cout<<endl;
 cout<<"max value="<<(pow(2,sizeof(double)*8.0-1)-1)<<endl;
}
```

Дана програма дозволяє переглянути характеристики типів даних у вашій системі. Оператор (операція) `sizeof()` обчислює кількість байт, що відводиться під тип даних або змінну. Функція `pow(x, y)` підносить значення `x` до степіня `y`, дана функція доступна у заголовному файлі <`cmath`>. Маніпулятори `fixed` і `setprecision()` доступні у заголовному файлі <`iomanip`>. Перший маніпулятор - `fixed`, передає в потік виводу (виводить на екран) дійсні значення в форматі з фіксованою комою. Маніпулятор `setprecision(n)` відображає `n` знаків після коми.

1.10 Контрольні запитання

1. Яку структуру має програма на С++?
2. Що таке препроцесор? Синтаксис директив препроцесора?
3. Що таке заголовний файл? Як він підключається?
4. Які оператори введення/виведення в С?
5. Як організоване введення/виведення в С++?
6. Які функції треба використовувати для введення/виведення кириліці?
7. Що таке тип даних?
8. Які існують базові типи в С++? Їх основні характеристики?
9. Що таке змінна? Синтаксис опису змінної?
10. Яким чином можна присвоїти змінній початкове значення?
11. Що таке константа?
12. Які різновиди констант є в мовіС++?
13. Що таке вираз?
14. Що таке операція в С++?
15. Які типи операцій визначені в С++?
16. Що таке операнд?
17. Яким чином визначається порядок обчислення виразу?
18. Який тип є результатом обчислення виразу?
19. Скільки існує рівней пріоритетів операцій С++?
20. Яку дію виконують операції інкремента та декремента?
21. Які бувають операції інкремента/декремента?
22. В чому різниця між постфіксною та префіксною формами операцій інкремента/декремента?
23. Яку дію виконує операція присвоєння?
24. Що робить складений оператор присвоєння?

25. Які в C++ є складені оператори присвоєння?

26. Які основні функції визначені в заголовному файлі `cmath`?

27. Як можна визначити розмір та максимальні значення типу даних в C++?

1.11 Задачі

1. Оголосити змінні, необхідні для обчислення площі прямокутника.

Результат:

```
float a, b; // ширина та довжина прямокутника
float s; // площа прямокутника
```

2. Оголосити змінні, необхідні для перерахунку ваги з фунтів у кілограми.

3. Оголосити змінні, необхідні обчислення площі кола.

4. Визначити вихідні дані та оголосити змінні, необхідні для обчислення % нарахування кредиту.

5. Оголосити змінні, необхідні обчислення вартості купівлі кількох транзисторів, силових кабелів.

6. Записати у вигляді інструкції надання формулу обчислення значення функції y = -2,7x3+ 0,23x2- 1,4.

Результат:

```
y:=-2.7*x*x*x + 0.23*x*x - 1.4;
```

7. Вивести на екран число π з точністю до сотих.

8. Три конденсатори зєднанні як показано на рисунку 1.2. Їх екввівлентра ємність дорівнює: $C=C_1+C_2C_3/C_2+C_3$. Значення C_1, C_2, C_3 вводить користувач з клавіатури.

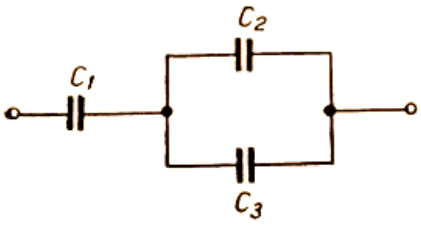

Рисунок 1.3 – Схема зєднання конденсаторів

9. Три конденсатори зєднанні як показано на рисунку 1.2. Їх екввівлентра ємність дорівнює: $C=C_1+C_2+C_3$. Значення C_1, C_2, C_3 вводить користувач з клавіатури.

10.Вивести на екран в один рядок числа 6, 23 и 49 з одним пробілом між ними.

11.Вивести на екран в один рядок числа 9, 33 и 72 з двома пробілом між ними.

12. Вивести на екран вираз для напруги електричного поля точок, що знаходяться на відстані $R \le R_0$ від центра шара. Абсолютна діелектрична проніцаємість середовища дорівнює ε_0, якщо:

а) $E = \frac{\rho}{3\varepsilon_0} \cdot R$.

б) $E = \frac{\rho \cdot R_0^3}{3\varepsilon_0 \cdot R^2}$

в) $E = \frac{\rho \cdot 4\pi \cdot R_0^3}{3\varepsilon_0 \cdot R^2}$

г) $E = \frac{\rho}{4\pi\varepsilon_0} \cdot R$

13.Відокремлений порожній металевий шар із зовнішнім радіусом R має надлишковий заряд Q. Ззсередини та ззовні сфери – вакуум. Чому дорівнює напруга електричного поля в центрі шара, якщо:

а) $E=\infty$. б) $E=0$ в) $E = \frac{Q}{4\pi\varepsilon_0 \cdot R^2}$

14. Записати у вигляді інструкції присвоювання формулу для обчислення струму за відомими значеннями напруги та опору електричного кола, що вводяться з клавіатури.

15.Записати у вигляді інструкції надання формулу обчислення опору електричного ланцюга за відомими значеннями напруги та сили струму, що вводяться з клавіатури.

16.Записати у вигляді інструкції надання формулу обчислення опору електричного ланцюга, що складається з трьох послідовно з'єднаних резисторів. Значення резисторів вводяться з клавіатури.

17. Записати у вигляді інструкції надання формулу обчислення опору електричного ланцюга, що складається з двох паралельно з'єднаних резисторів.

18.Записати у вигляді інструкції надання формулу перерахунку опору електричного ланцюга з омів у кіломи.

19. Вказати значення величини x після виконання наступних операторів присвоєння:

а) *x:*= 10; б) *x:*=- 10; в) *x:*= 60; в) *x:*=x- 1; г) *x:*= 0.

19. Скласти програму:

а) обчислення значення функції: $y = 7x^2 - 3x + 6$ значення x ввести з клавіатури;

б) обчислення значення функції: $x = 12a^2 + 7a - 16$ значення x ввести з клавіатури.

20. Визначити напругу між двома концетричними кульовими поверхнями, простыр між якими заповнене рівномірно зарядженим діелектриком з об'ємною щільністю заряду. Радіус поверхонь R_1(внутрішній) та R_2, абсолютна діелектрична принижність середовища:

а) $U = \frac{\rho}{3\varepsilon_\alpha}(R_2^2 - R_1^2)$;

б) $U = \frac{\rho}{6\varepsilon_\alpha}(R_1^2 - R_2^2)$;

в) $U = \frac{\rho R_2}{6\varepsilon_\alpha}(R_2 - R_1)$;

г) $U = \frac{\rho R_1}{6\varepsilon_\alpha}(R_2 - R_1)$

Приклад. Дано кут в градусах. Обчислити його косинус та синус

Лістінг 1.9

```
#include <cstdio> //Підключення бібліотеки вводу-виводу
#include <cmath> //Підключення бібліотеки математичних
функцій
using namespace std;
 int main() //Головна функція програми
 { //Оголошення та введення з клавіатури значення
   // змінної AlphaGrad
   int AlphaGrad
   printf("Введіть градусну міру кута alpha \n");
   scanf("%i", &AlphaGrad)
   // Переклад з градусної міри кута в радіанну
   double AlphaRad = AlphaGrad * M_PI / 180;
   // Обчислення косинуса й синуса кута
   double CosAlpha = cos(AlphaRad);
   double SinAlpha = sin(AlphaRad);
   // Виведення на екран косинуса й синуса кута
   printf("sin(%d) = %g, cos(%d) = %g", AlphaGrad, CosAlpha,
AlphaGrad);
```

1.12 Методичні вказівки

1. Для введення та виведення даних виористовувати операції >> , << та стандартні потоки `cin` і `cout`.

2. Для обчислення степеня можна використовувати функцію `pow(x,y)` з бібліотечного файла `cmath`.

2 Оператори С++

Мета: Отримання навичок у виборі і використанні операторів С ++; знайомство з ітераційними процесами.

2.1 Короткі теоретичні відомості

Оператори задають певну послідовність дій компілятора, але на відміну від виразів, є закінченими реченнями мови. Зазвичай оператори закінчуються крапкою з комою. Оператори мають назви, що відображають їх призначення або здійснювані дії.

Формат. Один оператор може займати один чи більше рядків. Два або більша кількість операторів можуть розташовуватися на одному рядку.

Вкладеність. Оператори, які управляють порядком виконання (`if, if - else, switch, while, do - while, for`) можуть бути вкладені один в один .

Мітка оператора - може стояти перед будь-яким оператором, щоб на цей оператор можна було перейти за допомогою оператора `goto`. Мітка складається з ідентифікатора, за яким стоїть двокрапка. Областю визначення мітки є даний блок.

```
AB2: x = 2;
```

Оператори поділяться на:

• порожній оператор;
• оператор-вираз;
• оператори оголошення;
• оператори присвоєння;
• оператори передачі управління (переходу);
• оператори вибору (розгалуження);
• оператори циклів.

Складений оператор (блок) складається з одного або більшої кількості операторів будь-якого типу, укладених у фігурні дужки {}. Після закриття фігурної дужки не повинно бути крапки з комою.

```
{x = 1; y = 2;}
```

Складені оператори і оператори, отримані в результаті вкладення (суперпозиції) операторів називаються *складними*.

До складних операторів відносять власне складові оператори і блоки. В обох випадках це послідовність операторів, укладена у фігурні дужки. Блок відрізняється від складеного оператора наявністю визначень в тілі блоку, наприклад:

```
{ n++;
  summa+=n; //це складений оператор
}
```

```
{ int n=0;
  n++; //це блок
  summa+=n;
}
```

Порожній оператор; використовується, коли по синтаксису оператор потрібен, а за змістом - ні.

Оператор - вираз: `a--;` - постфіксний інкремент; `fclose (file)` - виклик функції.

Оператор присвоєння: `y = (a * x + b) * x + c;` (також це операція присвоєння!).

Оператори оголошення - в яких оголошуються константи, змінні, заголовки функцій і типи даних: `int a;`

2.2 Оператори розгалуження (вибору)

Оператори вибору - це умовний оператор і перемикач. Умовний оператор має повну і скорочену форму.

2.2.1 Умовний оператор `if`

Умовний оператор `if` використовується для розгалуження процесу обчислень на два напрями (рис.2.1).

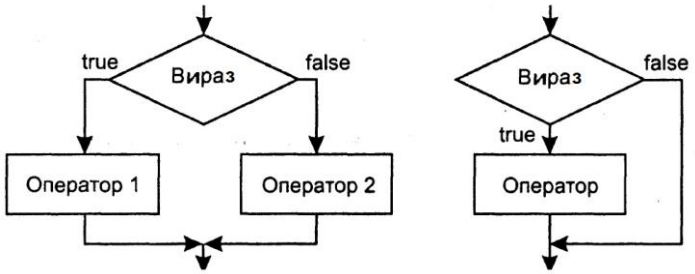

Рисунок 2.1 - Блок-схема умовного оператора: а) повна форма; б) скорочена форма

Формат оператора:

```
if (вираз-умова) оператор_1; [else оператор_2;]
```

Як `вираз-умова` можуть використовуватися арифметичний вираз, відношення і логічний вираз. Спочатку обчислюється `вираз`, який може мати арифметичний тип або тип вказівник. Якщо він не дорівнює нулю (має значення `true`), виконується перший оператор, інакше - другий. Після цього управління передається на оператор, наступний за умовним. Якщо значення `виразу-умови` відмінне від нуля (тобто `Істина`), то виконується `оператор_1`, якщо значення `виразу-умови` дорівнює нулю (тобто `Хибне`), то виконується `оператор_2` наприклад:

```
        if (x<y&&x<z) min=x;
        //якщо x<y та x<z змінна набуває min значення x

        if (d>=0)
        {    x1=(-b-sqrt(d))/(2*a);
             x2=(-b+sqrt(d))/(2*a);
             cout<< "\nx1="<<x1<<"x2="<<x2;
        }
             else cout<<"\nРозв'язку немає";
// якщо дискримінант більше або дорівнює 0,
// обчислюємо корені квадратного рівняння
```

Одна з гілок може бути відсутня, логічніше опускати другу гілку разом з ключовим словом `else`. Якщо в будь-якій гілці потрібно виконати декілька операторів, їх необхідно укласти в блок, інакше компілятор не зможе зрозуміти, де закінчується розгалуження. Блок може містити будь-які оператори, в тому числі опису та інші умовні оператори (але не може складатися з одних описів). Необхідно враховувати, що змінна, описана в блоці, поза блоком не існує. Оператори можуть бути вкладені один в один.

Приклади:

```
if (a<0) b=1;                                                    //1
if (a<b && (a>d || a==0))b++; else {b*=a; a=0;}                  //2
if (a<b) {if (a<c) m=a; else m=c;} else {if (b<c) m=b;
                                          else m=c;}             //3
if (a++) b++;                                                    //4
if (b>a) max =a;
     else max=a;                                                 //5
```

В 1 відсутня гілка `else`.

Якщо потрібно перевірити кілька умов, їх об'єднують знаками логічних операцій. Наприклад, вираз в 2 буде істиним в тому випадку, якщо виконається одночасно умова `a<b` і одна з умов в дужках. Якщо опустити внутрішні дужки, буде виконано спочатку логічне `I`, а потім - `АБО`.

Оператор в 3 обчислює найменше значення з трьох змінних. Фігурні дужки в даному випадку не є обов'язковими, так як компілятор відносить частину `else` до найближчого `if`.

4 демонструє те, що хоча в якості виразів в операторі `if` найчастіше використовуються операції відношення, це не обов'язково.

Конструкції, подібні оператору 5, простіше і наочніше записувати у вигляді умовної (тернарной) операції (в даному випадку:

```
max = (b>a) ? b : a;
```

так як одержувані при використанні умовної операції вирази більш компактні і ведуть до отримання більш компактного коду.

!!! Поширена помилка при запису умовних операторів - використання в виразах замість перевірки на рівність (==) простого присвоєння (=), наприклад,

```
if(a=1) b=0;
```

Синтаксичної помилки немає, так як операція присвоєння формує результат, який і оцінюється на еквівалентність 0. В даному прикладі присвоєння змінній `b` буде виконано незалежно від значення змінної `a`. Тому в виразах перевірки змінної на рівність константі рекомендується записувати константу зліва від операції порівняння:

```
if (1==a) b=0;.
```

При порівнняні змінної на рівність/нерівність 0, замість виразів:

```
(a==0) // дорівнює нулю
(a!=0) // не дорівнює нулю
```

краще використувавати наступний запис:

```
(!a) // дорівнює нулю
(a)  // не дорівнює нулю
```

Обидва записи семантично вірні, але з точки зору використання можливостей синтаксису C++ другий формат краще, і є ознакою хорошого стилю.

!!!Друга помилка - *невірний запис перевірки на приналежність діапазону*. Наприклад, щоб перевірити умову `0<x<1`, не можна записати її в умовному операторі безпосередньо у такому вигляді, так як буде виконано спочатку порівняння `0<x`, а його результат (`true` або `false`, перетворене в `int`) буде порівнюватися з 1. Правильний запис:

```
if (0 <x && x <1) ...
```

Якщо яка-небудь змінна використовується тільки всередині умовного оператора, рекомендується об'ява її всередині дужок, наприклад:

```
if (int i = fun (t)) a -= i; else a += i;
```

Оголошення змінної в той момент, коли вона потрібна, тобто коли їй необхідно присвоїти значення, є ознакою хорошого стилю і дозволяє уникнути випадкового використання змінної до її ініціалізації. Оголошувати всередині оператора `if` можна тільки одну змінну. Область її видимості починається з точки оголошення і включає обидві гілки оператора.

2.2.2 Оператор перемикач

Оператор `switch` (перемикач) призначений для розгалуження процесу обчислень на кілька напрямків. Формат оператора:

```
switch (вираз)
{  case константний_вираз_1: [список_операторів_1]
   case константний_вираз_2: [список_операторів_2]
   ...
   case константний_вираз_3: [список_операторів_n]
   [default: оператори]
}
```

Виконання оператора починається з обчислення виразу (він повинний бути цілочисельним), його значення послідовно порівнюється з константами, які записані слідом за `case`, і управління передається оператору зі списку, позначеному константним виразом, значення якого співпало з обчисленим. Після цього, **якщо вихід з перемикача явно не вказаний, послідовно виконуються всі інші гілки.** Вихід з перемикача зазвичай виконується за допомогою операторів `break` або `return`. Якщо значення виразу, записаного після `switch`, не співпало з жодною константою, то виконуються оператори, які слідують за міткою `default`. Мітка `default` може бути відсутня.

Всі константні вирази повинні мати різні значення, але бути одного і того ж цілочисельного типу. Кілька міток можуть слідувати поспіль. Якщо збігів не відбулося, виконуються оператори, розташовані після слова `default` (а при його відсутності управління передається наступному за `switch` оператору). У разі синтаксичної помилки в слові `default` повідомлення про помилку не видається, оскільки компілятор сприйме це слово як допустиму позначку (мітку) оператора. Досвідчені програмісти для пошуку помилок часто включають гілку `default`, навіть коли враховані всі можливі варіанти.

```
// Лістинг 2.1. Найпростіший калькулятор на 4 дії
int main ()
{  int a, b, res;
   char op;
   cout << "\nВведіть перший операнд:"; cin >> a;
   cout << "\nВведіть знак операції:" ; cin >> op;
   cout << "\nВведіть другий операнд:"; cin >> b;
   bool f = true;
   switch (op)
   { case '+': res = a + b; break;
     case '-': res = a - b; break;
     case '*': res = a * b; break;
     case '/': res = a / b; break;
     default: cout <<"\nНевідома операція"; f = false;
   }
   if (f) cout <<"\nРезультат:" << res;
}
```

Вводимо перший операнд (змінна `a` типа `int`), знак операції `op` та другий операнд (змінна `b` типа `int`). В залежності від значення `op` обирається одна з чотирьох математичних дій та виконуються додавання, віднімання, множення або ділення операндів відповідно. Якщо введений якийсь інший операнд, виводиться про це повідомлення, та змінній `f` присвоюється значення `false`. Після оператора аналізується значення `f` , якщо воне істинно на екран виводиться результат обчислення.

2.3 Оператори цикла

Цикл являє собою ділянку програми, в якому одні й ті ж дії реалізуються неодноразово над різними значеннями одних і тих же змінних (об'єктів).

2.3.1 Цикл з передумовою (`while`)

Використовується в разі, коли по-перше, невідома заздалегідь кількість повторень, і, по-друге, при цьому немає необхідності, щоб цикл неодмінно був виконаний хоча б один раз. Цикл з передумовою реалізує структурну схему (рис.2.2), і має синтаксис:

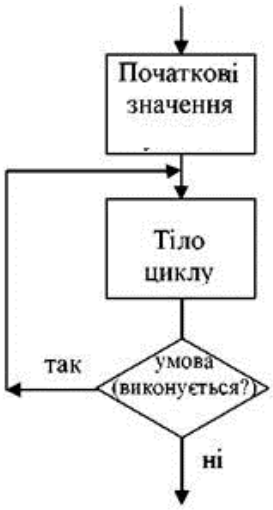

Рисунок 2.2 - Блок-схема циклу з передумовою

```
while (вираз-умова) тіло_цикла;
```

Вираз-умова визначає умову повторення тіла_цикла, представленого простим або складеним оператором. Виконання оператора починається з обчислення виразу. Якщо воно істинне (не дорівнює false), виконується тіло циклу. Якщо при першій перевірці вираз дорівнює false, цикл не виконається жодного разу. Тип виразу-умови повинен бути арифметичним (логічним) або приводиться до нього. Вираз обчислюється перед кожною ітерацією циклу. Тіло циклу виконується до тих пір, поки вираз-умова не стане хибним.

При реалізації циклу в його тіло обов'язково повинні бути включені конструкції, які змінюють логіку виразу, яке перевіряється, так, щоб врешті-решт, він став хибним. Інакше цикл буде нескінченним.

```
//Лістинг 2.2. Знаходження всіх дільників
// цілого позитивного числа за допомогою цикла while
int main()
{
     int num;
     cout <<"\nВведіть число:";
     cin >> num;
     int half=num/2;          // половина числа
     int div=2;               // кандидат на дільник
     while (div<=half)
     {
          if (!(num % div))
               cout << div <<"\n";
          div++;
     }
}
```

Перед початком циклу змінним half та div присвоюються початкові значення. Якщо умова істинна (div<=half), виконується тіло циклу, в якому змінюється значення змінної div, що призводить до зміни логіки виразу-умови. Обчислення виконується до тих пір, поки кандидат на дільник div не стане завбільшки ніж половина числа. В тілі цикла виконується перевірка чи є кандидат div дільником числа (якщо остача від ділення дорівнює 0).

В круглих дужках після ключового слова while можна вводити опис змінної. Областю її дії є цикл:

```
while (int x = 0) {... / * область дії x * /}
```

Поширений прийом програмування - організація нескінченного циклу з заголовком while (true) або while (1) і примусовим виходом з тіла циклу за виконання певної умови.

2.3.2 Цикл з постумовою (`do while`)

Використовується в тому разі, коли заздалегідь невідома кількість повторень, але при цьому необхідно виконати цикл хоча б один раз. Цикл з постумовою реалізує структурну схему (рис.2.3) і має синтаксис:

Рисунок 2.3 - Блок-схема цикла з післяумовою

```
do тіло_цикла while вираз-умова;
```

Спочатку виконується простий або складений оператор, що складає `тіло_циклу`, а потім обчислюється `вираз_умова`. Якщо він істинний (не дорівнює `false`), `тіло_циклу` виконується ще раз. Цикл завершується, коли `вираз-умова` стане рівним `false` або в тілі циклу буде виконаний будь-який оператор передачі управління. Тип виразу повинен бути арифметичним або приводиться до нього.

```
//Лістинг 2.3. Програма здійснює перевірку введення:
int main ()
{  char answer;
     do
     {    cout << "\nКупи слоненятко!";
          cin >> answer;
     }
     while (answer!='y');
}
```

Поки користувач не введе `y`, на екрані буде з’являтися напис `Купи слоненятко!`

2.3.3 Цикл із параметром (`for`)

Цикл з параметром має наступний формат:

```
for (ініціалізація; вираз; модифікація) тіло_цикла;
```

`Ініціалізація` використовується для оголошення і присвоєння початкових значень величинам використовуваним в циклі. У цій частині можна записати кілька операторів, розділених комою (операцією "послідовне виконання"), наприклад так:

```
for (int i = 0, j = 2; ...;)
int k, m;
for (k = 1, m = 0; ...;)
```

Областю дії змінних, оголошених в частині ініціалізації циклу, є цикл. Ініціалізація виконується один раз на початку виконання циклу.

Вираз визначає умову виконання циклу: якщо його результат, приведений до типу bool, дорівнює true, цикл виконується. Цикл із параметром реалізований як цикл з передумовою.

Модифікації виконуються після кожної ітерації циклу і служать зазвичай для зміни параметрів циклу. У частині модифікації можна записати кілька операторів через кому. Простий або складний оператор являє собою тіло_циклу.

```
//Приклад (оператор обчислює суму чисел від 1 до 100)
s=0;
for (int i= 1; s<=100; i++) s+=i;           //1
```

можна так:

```
for (int i=1, s=0; s<=100; i++) s+=i;       //2
```

а можна і так:

```
for (int i=1, s=0; s<=100; s+=i, i++);      //3
```

і навіть так:

```
for (int i=1, s=0; s<=100; s+=i++);         //4
```

В 1 прикладі початкове значення суми задається перед циклом, у 2 в частині ініціалізації циклу. У 3 обчислення суми внесено в частину модифікації циклу перед збільшенням лічильника, а в 4 два оператора в частині модифікації об'єднані в один. Зверніть увагу, що в 3 і 4 тіло циклу складається з порожнього оператора;.

Будь-яка з частин оператора for може бути опущена (але крапки з комою треба залишити на своїх місцях). Якщо опущений вираз, то вважається, що він постійно істинний (true).

Оператор for (;;); являє собою нескінченний цикл, еквівалентний оператору while (1).

Будь-який цикл while може бути приведений до еквівалентного йому циклу for і навпаки за наступною схемою:

```
for (b1;b2;b3) оператор
```

```
b1;
while (b2)
  {оператор; b3;}
```

```
//Лістинг 2.4. Находження всіх дільників цілого позитивного
//числа за допомогою цикла for
#include <iostream>
int main()
{    int num, half, div;
     cout <<"\nВедіть число:";
     cin >> num;
     for (half=num/2,div=2; div<=half; div++)
          if (!(num % div))
               cout << div <<"\n";
}
```

У цьому лістингу наведена реалізація приклада з лістингу 2.2 за допомогою цикла `for`. Початкові значення змінним `half` та `div` присвоюються у заголовку цикла у частині ініціалізації, зміна значення `div` також відбувається у заголовку цикла у частині модифікації. Якщо порівняти коди у лістингах 2.2 та 2.4 стає очевидним, що всі зміни відповідають наведеній схемі перетворення цикла `while` в цикл `for`.

Можливості цикла `for:`

1. Можна застосовувати операцію зменшення для рахунку в порядку убування.

Приклад. Потрібно обчислити y^5. Можливе рішення має вигляд:

```
for ( i=5, r=1; i>=1; i--) r=r*y;
 cout<<"r="<<r;
```

2. При бажанні можна організувати рахунок двійками, трійками, десятками і т.д.

Приклад. Приріст рахунку, відмінний від 1.

```
for ( n=5; n<61; n+=15) cout<<n;
```

3. Можна в якості лічильника використовувати не тільки цифри, але і символи.

Приклад. Потрібно надрукувати алфавіт. Можливе рішення має вигляд:

```
for ( chr='A'; chr<='Z'; chr++) cout<<chr;
```

4. Можна задати зростання значень лічильника не в арифметичній, а в геометричній прогресії.

Приклад. Потрібно підрахувати борг. Можливе рішення має вигляд:

```
for ( k=100; k<185; k*=1.1) cout<<"Долг="<<k;
```

5. В якості третього вираза можна використовувати будь-який правильно складений вираз. Він буде обчислюватися в кінці кожної ітерації.

Приклад. Використання в якості лічильника вираза.

```
for (k=1; z<=196; z=5*k+23 ) cout<<z;
```

6. Можна пропускати один або кілька виразів. (При цьому не можна пропускати символи `;` - крапка з комою.)

Приклад. Неповний список виразів в заголовку тіла циклу.

```
for (p=2; p<=202;) p=p+n/k;
```

7. Перший вираз не обов'язково має ініціювати змінні, він може бути будь-якого типу.

Приклад. Довільний перший вираз в заголовку циклу.

```
for (cout<<"Введіть числа."; p<=30;)
     cin>>p;
```

8. Змінні, що входять до виразів специфікації циклу, можна змінювати в тілі циклу.

Приклад. Зміна управляючих змінних в тілі циклу.

```
delta=0.1;
for (k=1; k<500; k+=delta)
     if (a>b) delta=0.5;
```

9. Використання операції , (кома) в специфікації циклу дозволяє включати кілька ініціюючих і коригувальних виразів.

Приклад. Використання операції , (кома) в специфікації циклу.

```
for (i=1, r=1; i<=10; i++, r*=y)
     cout<<"y в степені "<<i<<"="<<r;
```

10. Можна перевіряти будь-яку іншу умову, яка відрізняється від кількості ітерацій.

Приклад. Умова виходу - куб лічильника циклу дорівнює або більше 216.

```
     for ( num=1; num*num*num<216; num++);
```

2.3.4 Вкладені цикли

Вкладеним циклом називають конструкцію, в якій один цикл виконується всередині іншого. Внутрішній цикл виконується повністю під час кожної ітерації зовнішнього циклу.

```
//Лістинг 2.5. Вкладені цикли. Заповнення всього екрана
// символами '#'.
for (i=l; i<=25; i++)
     for (k=l; k<=80; k++)
          cout<<'#';
```

У даній програмі 25 раз здійснюється виведення на екран по 80 символів '#'.

У програмі можна використовувати будь-які комбінації вкладених циклів всіх типів: `while, for, do - while`, якщо цього вимагає логіка побудови програми.

```
//Лістинг 2.6. Здійснити введення десяти значень днів
// місяця з перевіркою правильності введення.
int den;
for (i=l; i<=10; i++)
{    do cin>>den;
     while (den<l||den>31);
     cout<<den;
}
```

В даному прикладі зовнішній цикл виконується 10 разів, а внутрішній буде виконуватися до тих пір, поки не буде введено вірне значення.

```
//Лістинг 2.7. Друк натуральних чисел від 1 до 15
//по п'ятьох стовпчиках.
int main ()
{  int i,j;
    for (i=1; i<=3; i++)
    { for (j=1; j<=5; j++)
        cout << 3*(j-1)+i << " ";
      cout << "\n";
    }
}
```

Результат виконання програми буде виглядати так:

```
1 4 7 10 13
2 5 8 11 14
3 б 9 12 15
```

Індексна змінна `i` зовнішнього оператора циклу приймає значення від 1 до 3 включно. Ця змінна визначає номер рядка, в якому відображається число. Номер стовпця визначається індексною змінною `j` внутрішнього оператора циклу. Ця змінна при кожному фіксованому значенні змінної `i` пробігає значення від 1 до 5. У внутрішньому операторі циклу виконується всього одна команда

```
cout << 3*(j-1)+i << " ";,
```

за допомогою якої роздруковуються числа у відповідному рядку. Що стосується зовнішнього оператора циклу, то в рамках кожного циклу виконується дві команди: внутрішній оператор циклу, в якому роздруковуються числа в рядку, і команда `cout<<"\n";` (для переходу на новий рядок).

Слід чітко розуміти різницю між двома (або більше) вкладеними операторами циклу і одним оператором циклу з декількома індексними змінними.

```
//Лістинг 2.8. Один оператор циклу з декількома
//індексними змінними
int main ()
{   int i,j;
    for (i=10,j = 90; i<j; i+=5,j-=10)
       cout << i <<” ” << j <<”\n”;
}
```

Результатом виконання програми буде два стовпця чисел:

```
10 90
15 80
20 70
25 60
30 50
35 40
```

На відміну від випадку вкладених операторів, в даній ситуації обидві індексні змінні `i` та `j` змінюються синхронно. Змінна `i` ініціалізується зі значенням 10, а змінна `j` - значенням 90. За кожен цикл значення змінної `i` збільшується на 5, а значення змінної `j` зменшується на 10. Значення змінних виводяться на екран. Процес триває до тих пір, поки значення змінної `i` менше значення змінної `j`.

2.3.5 Рекомендації щодо застосування циклів

Найчастіщі **помилки програмування циклів** - використання в тілі циклу неініціалізованих змінних і невірний запис умови виходу з циклу.

Щоб уникнути помилок рекомендується:

• перевірити, чи всім змінним, які зустрічаються в правій частині операторів присвоєння в тілі циклу, присвоєні до цього початкові значення;

• перевірити, чи змінюється в циклі хоча б одна змінна, що входить в умову виходу з циклу;

• передбачити аварійний вихід з циклу по досягненню деякої кількості ітерацій;

• і звичайно, не забувати про те, що якщо в тілі циклу потрібно виконати більше одного оператора, потрібно укладати їх в фігурні дужки.

Оператори циклу взаємозамінні. Але можна навести деякі рекомендації по вибору найкращого в кожному конкретному випадку:

- Оператор `do while` зазвичай використовують, коли цикл потрібно виконати хоча б раз.
- Оператором `while` зручніше користуватися у випадках, коли число ітерацій заздалегідь невідомо, очевидних параметрів циклу немає.
- Оператор `for` краще в більшості інших випадків (однозначно для організації циклів з лічильниками). В С ++ він є більш гнучким засобом, ніж аналогічні оператори циклів в інших мовах програмування.

2.4 Оператори передачі управління

В С ++ є чотири оператори, що змінюють природний порядок виконання обчислень:

• оператор виходу з циклу `break`;

• оператор переходу до наступної ітерації циклу `continue`;

• оператор повернення з функції `return`.

• оператор безумовного переходу `goto`;

Оператор завершення `break` - припиняє виконання найближчого вкладеного зовнішнього оператора `switch, while, do, for`.

Управління передається оператору, наступному за тим, що закінчується. Оператор `break` доцільно використовувати, коли умову продовження ітерацій треба перевіряти в середині циклу.

Приклад:

```
// Лістинг 2.9. Використання оператора break.
// Якщо елемент масиву дорівнює нулю -
// вихід з циклу
 for (i = 0; ; i++)
      if ((a[i]=b[i])==0) break;
```

В заголовку цикла немає умови виходу з цикла, в тілі цикла відбувається копіювання значення `b[i]` в `a[i]`, і якщо значення `b[i]` дорівнює 0, виконання циклу переривається оператором `break`.

```
// Лістинг 2.10. Знаходження суми чисел, числа вводяться з
// клавіатури до тих пір,
// поки не буде введено 100 чисел або 0
     for(s=0, i=1; i<100; i++)
     { cin>>x;
       if( x==0) break; //краще if(!x)
          //якщо ввели 0, то підсумовування закінчується
       s+=x;
     }
```

Оператор продовження `continue` - передає управління на початок найближчого зовнішнього оператора циклу `while`, `do`, `for`, викликаючи початок наступної ітерації. Цей оператор за дією протилежний оператору `break`. Використовується тільки в тілі циклу.

```
// Лістинг 2.11. Використання оператора continue.
// Якщо елемент масиву не дорівнює нулю -
// перехід до наступної ітерації циклу
 for (i=0; i++;)
 {   if (a[i]!=0) continue;
     a[i]=b[i];
     k++;
}
```

Цей приклад виконує тіж дії, що і фрагмент у лістингу 2.9, але якщо значення `a[i]` не дорівнює 0, переходимо на наступну ітерацію циклу.

```
// Лістинг 2.12. Використання оператора continue.
// Знаходження кількості і суми позитивних чисел
 for( k=0,s=0,x=1;x!=0;)
     { cin>>x;
       if (x<=0) continue;
       k++;s+=x;
     }
```

Якщо введене значення `x` менше або дорівнює 0 – перехід на наступну ітерацію цикла без додавання значення `x` в суму.

Використання оператора `continue` не дуже доцільне, можна організувати обчислювальний процес таким чином, щоб уникнути непотрібних порівнянь та зайвих дій.

Оператор повернення `return` – припиняє виконання поточної функції і повертає управління програмі, яка викликала функцію.

```
return вираз; // з передачею значення виразу.
return; // без передачі значення виразу.
```

Приклад: `return x + y;`

```
// Лістинг 2.13. Опис та виклик функції,
//яка повертає суму двох чисел
#include <iostream.h>
int sum (int, int);

void main()
{  cout<<sum(4,3); //виклик функції }

int sum (int a, int b) //визначення функції
{   return (a+b);  }    //повернення значення
```

Оператор безумовного переходу - передає управління на оператор з міткою. Оператор безумовного переходу `goto` має формат:

```
goto мітка;
```

Використовується для виходу із вкладених управляючих операторів. Область дії обмежена поточною функцією. У тілі тієї ж функції повинна бути присутня рівно одна конструкція виду:

```
Мітка: оператор;
```

`Мітка` - це звичайний ідентифікатор, область видимості якого є функцією, в тілі якої він заданий. Приклад: `goto ABC`;

Використання оператора безумовного переходу виправдано в двох випадках:

1. Примусовий вихід вниз по тексту програми з декількох вкладених циклів або перемикачів.

2. Перехід з декількох місць функції в одне (наприклад, якщо перед виходом з функції завжди необхідно виконувати будь - які дії).

В інших випадках для запису будь-якого алгоритму існують більш придатні засоби, а використання `goto` призводить тільки до ускладнення структури програми і утруднення налагодження. Застосування `goto` порушує принципи структурного і модульного програмування, за якими всі блоки, з яких складається програма, повинні мати тільки один вихід і один вхід.

У будь-якому разі не слід передавати управління всередину операторів `if, switch` і циклів. Не можна переходити всередину блоків, що містять ініціалізацію змінних, на оператори розташовані після неї, оскільки в цьому випадку ініціалізація НЕ буде виконана:

```
int k; …
goto Metka; …
{   int a = 3, b = 4;
    k = a + b;
    Metka: int m = k + 1 …
}
```

Після виконання цього фрагмента програми значення змінної `m` не визначене.

2.5 Контрольні запитання

1. Що таке оператор? В чому полягає його відмінність від операції та виразу? Які бувають оператори?
2. Призначення умовного оператор?
3. Назвіть дві форми умовного оператора?
4. Навіщо використовують ключове слово «`else`»?
5. Чи можуть виконатися одночасно обидві гілки в умовному операторі?
6. Чи може хоч одна з гілок умовного оператора бути командою розгалуження?
7. Якою операцією можна замінити в деяких випадках умовний оператор?
8. Який синтаксис має оператор вибору?
9. Які особливості виконання оператора вибору?
10. У чому полягає відмінність між циклами з передумовою та циклами з постумовою?
11. Що може спричинити зациклювання програми?
12. За яких умов цикли `while` і `for` не виконуються жодного разу?
13. Як можна замінити цикл `while` на `for` і навпаки?
14. Яким чином можна перервати роботу циклу?
15. Що таке вкладені цикли?

2.6 Завдання

2.6.1 Умовний оператор

1. Написати програму обчислення опору електричного ланцюга, що складається з двох опорів, які можуть бути з'єднані послідовно або паралельно.

Далі наведено рекомендований вигляд екрана (дані, введені користувачем, виділені жирним).

Обчислення опору електричного ланцюга

Введіть вихідні дані:

Величина першого опору (Ом) -> 15

Величина другого опору (Ом) -> 27.3

Тип з'єднання (1-послідовне, 2-паралельне) -> 2

---

2. Відомі опори двох нез'єднаних один з одним ділянок електричного ланцюга та напруга на кожному з них, що вводяться з клавіатури. Якою ділянкою протікає менший струм?

3. Відомі дві одиниці вимірювання: одна за кулон, інша — за Ом, що вводяться щ клавіатури. Яка з одиниць вимірювання менша?

4. Дано обсяги та маси двох тіл з різних матеріалів. Матеріал якого з тіл має більшу щільність?

5. Відомо рік та номер місяця народження людини, а також рік та номер місяця сьогоднішнього дня (січень - 1 і т. д.). Визначити вік людини (число повних років). У разі збігу вказаних номерів місяців вважати, що

пройшов повний рік.

6. Дано значення опору. Визначити:

а) яке з значення більше; б) чи однакові значення.

7. Коаксіальний кабель з двошаровою ізоляцією знаходиться під напругою *U*. Оболонка кабеля заземлена. Визначити напругу електричного поля *E,* якщо відстань r=10, а $E = \frac{U}{r(\ln\frac{r_2}{r_1} + \frac{\varepsilon_1}{\varepsilon_2}\ln\frac{r_3}{r_2})}$.

8. . Коаксіальний кабель з двошаровою ізоляцією знаходиться під напругою *U*. Оболонка кабеля заземлена. Визначити напругу електричного поля *E,* якщо відстань r=45, а $E = \frac{U}{r(\ln\frac{r_2}{r_1} + \varepsilon_1\ln\frac{r_3}{r_2})}$.

9. Коаксіальний кабель з двошаровою ізоляцією знаходиться під напругою *U*. Оболонка кабеля заземлена. Визначити напругу електричного поля *E,* якщо відстань r=53, а $E = \frac{U}{r(\varepsilon_1\ln\frac{r_2}{r_1} + \varepsilon_2\ln\frac{r_3}{r_2})}$.

10. Визначити об'ємну щільність енергії електричного поля, якщо $E=2\times10^2$ *кВ/м;* $D = 2$ *мкКл/м*^{2}, $W=2\times10^{-1}$ *Дж/м*^{3}.

11. Визначити об'ємну щільність енергії електричного поля, якщо $E=1\times10^2$ *кВ/м;* $D = 1$ *мкКл/м*^{2}, $W=1{,}73\times10^{-1}$ *Дж/м*^{3}.

12. Визначити об'ємну щільність енергії електричного поля, якщо $E=1{,}2\times10^2$ *кВ/м;* $D = 2$ *мкКл/м*^{2}, $W=4\times10^{-1}$ *Дж/м*^{3}.

13. Плоский конденсатор має параметри: площа кожної пластини $S=20$ *см*^{2}, відстань між пластинами *d=5мм*, діелектрична проникність ізоляції $\varepsilon=5$. Знайти силу *F* взаємного при тяжіння пластин конденсатора, при умові що пластини були приєднанні до джерела з напругою *U=1 кВт*, якщо $F=2\times10^{10}$*Н.*

14. Плоский конденсатор має параметри: площа кожної пластини $S=20$ *см*^{2}, відстань між пластинами *d=5мм*, діелектрична проникність ізоляції $\varepsilon=5$. Знайти силу *F* взаємного при тяжіння пластин конденсатора, при умові що пластини були приєднанні до джерела з напругою *U=1 кВт*, якщо *F=128мН.*

15. Плоский конденсатор має параметри: площа кожної пластини $S=20$ *см*^{2}, відстань між пластинами *d=5мм*, діелектрична проникність ізоляції $\varepsilon=5$. Знайти силу *F* взаємного при тяжіння пластин конденсатора, при умові що пластини були приєднанні до джерела з напругою *U=1 кВт*, якщо *F=25,6мН.*

16. Плоский конденсатор має параметри: площа кожної пластини $S=20$ *см*^{2}, відстань між пластинами *d=5мм*, діелектрична проникність ізоляції $\varepsilon=5$. Знайти силу *F* взаємного при тяжіння пластин конденсатора, при умові що пластини були приєднанні до джерела з напругою *U=1 кВт*, якщо *F=0,12 мН.*

17. Визначити, які із трьох джерел, представлених на рисунку 2.4 генерують енергію, а які – вживають, якщо R_1=6 Ом, R_2=8 Ом, R_3=3 Ом, E_1=10В, E_2=30 Ом, E_3=30 Ом.

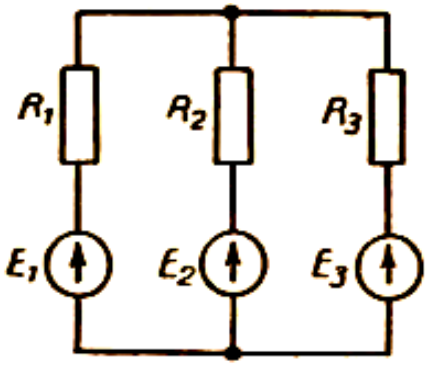

Рисунок 2.4 – Структурна схема зєднання джерел живлення

18. Є три числа `A, B, C`, знайти площу трикутника з довжиною сторін `A, B, C;` якщо площа трикутника дорівнює 0, то видати про це повідомлення.

19. За номером `y (y>0)` певного віку визначити `c` - номер його століття (врахувати, що, наприклад, початком XX століття був 1901, а не 1900 рік).

20. Обчислити `y=tg(x) (0<x<2); 1-tg(x)` інакше.

21. Обчислити `y=1 (x<0); y=cos(x)(0<x<π); y=-1 (x>π).`

22. Обчислити `y=-x (x<0); y=√x (x>0).`

23. Обчислити `y=ctg(x) (x<0); y=x`^{2}`-4 (x>0).`

24. Обчислити `y=|x| (x<1); y=1 (1<x<3); y=-x.`

25. Обчислити `y=sin(x) (x<1); y=x`^{3} `(x>1).`

26. Обчислити `y=|x| (x<0); y=x`^{3} `(x>0).`

27. Обчислити `y=max(x, y, a).`

28. Обчислити `y=min(x,y,z).`

29. Обчислити `y=e`^{x} `(x>0); y=x-1 (x <0).`

30. Обчислити `y=sin(x) (0 <X <2); 1-cos(x)` інакше.

Приклад. Написати програму для розв’язування квадратного рівняння $ax^2 + bx + c = 0$, скориставшись формулами $D = b^2 + 4ac$, $x_{1,2}=(-b \pm \sqrt{D})/2a$

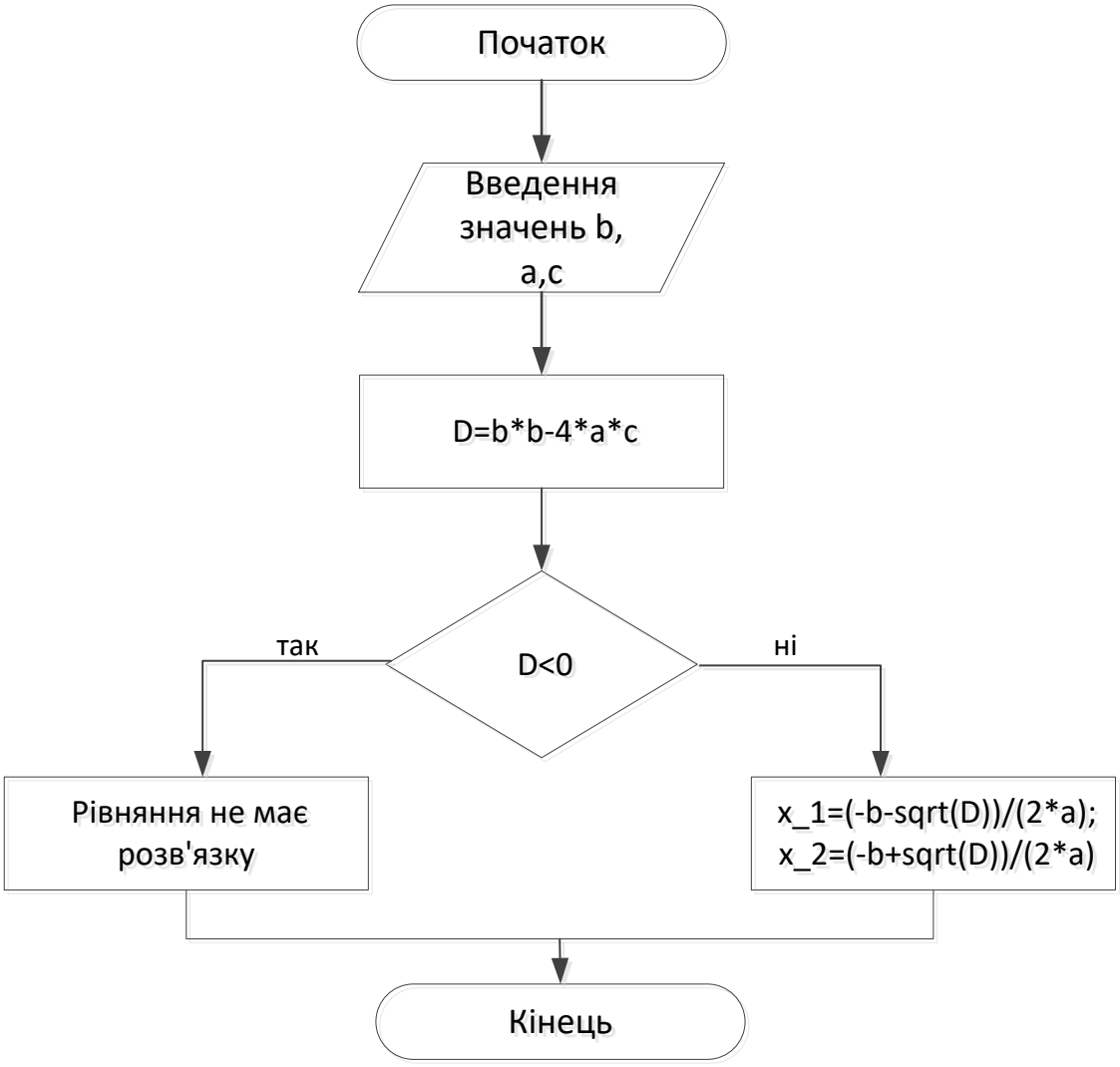

Рисунок 2.4 – Блок схема алоритму

Лістінг 2.14

```
/*Рівняння має розв’язок, якщо D ≥ 0 , і не має розв’язку при D < 0 .*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{ // Розв’язування квадратного рівняння
  double a,b,c,D,x_1,x_2;
  system("chcp 1251 & cls");
/* установка кодової сторінки win-cp 1251 (кодування ANSI) в потоки вводу і виводу (change codepage)*/
  printf("Розв'язування рівняння ax2+bx+c=0,c\n");
  printf("Введіть a,b,c\n\t");
  scanf("%lf%lf%lf",&a,&b,&c);
  D=b*b-4*a*c;
  if (D<0)
      printf("Рівняння не має розв'язку");
  else { x_1=(-b-sqrt(D))/(2*a);
         x_2=(-b+sqrt(D))/(2*a);
```

```
        printf("Розв'язок:\n");
        printf("\tx1=%0.3f; x2=%0.3f\n",x_1,x_2);
      }
    printf("\n\n");
    system("pause");
    return 0;
  }
```

2.6.2 Перемикач

1. Є ціле число. Якщо воно ділиться на 7 без остачі обчислити квадрат цього числа, із залишком 5 - синус цього числа, із залишком 3 - натуральний логарифм. Вивести результат на екран.

2. За номером однієї з п'яти європейських країн (на Ваш вибір) вивести назву столиці країни.

3. За номером однієї з чотирьох мов програмування (за Вашим вибором) вивести назву цієї мови програмування.

4. У відповідності зі значенням цифри вивести назву цієї цифри.

5. За номером ноти вивести її назву.

6. За вибором користувача вивести розмір одиниці вимірювання (см, дм, м, км) в мм.

7. За номером констант кольору вивести його назву.

8. Маючи дату (місяць і день народження) визначте знак Зодіаку.

9. Напишіть програму для введення номера певного віку нашої ери і виведення на екран його назви за східним календарем.

10. Напишіть програму для виведення на друк дня початку заданого місяця поточного року.

11. Напишіть програму для виведення на екран значень римських цифр.

12. За номером дня тижня вивести на екран його назву українською мовою.

13. За номером спеціальності (5 на вибір) вивести їх назву українською мовою.

14. За номером дня тижня вивести на екран його назву англійською мовою.

15. За номером місяця року вивести на екран його назву українською мовою.

16. За номером спеціальності (5 на вибір) вивести їх назву англійською мовою.

17. За номером місяця року вивести на екран його назву англійською мовою.

18. За номером функціональних клавіш в середовищі програмування вивести їх призначення.

19. За номером перших десяти латинських літер вивести їх назву.

20. За номером перших десяти українських літер вивести їх назву.

21. За номером перших десяти російських літер вивести їх назву.

22. Визначити `k` - порядковий номер того дня високосного року, який має дату, введену користувачем (день, місяць).

23. За номером місяця вивести кількість днів у цьому місяці.

24. За номером кольору у веселці вивести його назву.

25. За номером планети в Сонячній системі вивести її назву.

26. За номером поверха в корпусі 10 університету вивести назву кафедр, які знаходяться на цьому поверсі.

27. За номером (починаючи з мізинчика) вивести назву кожного пальця на руці.

28. За номером курсу вивести його назву (перший, другий ...).

29. За номером місяця вивести на екран назву сезону, до якого відноситься місяць.

30. За номером місяця вивести кількість днів у ньому (рік не високосний!).

2.6.3 Оператори цикла

(завдання повинно бути виконано всіма трьома типами циклів)

1. Знайти середнє арифметичне цілих чисел від 1 до n.

2. Знайти середнє геометричне цілих чисел від 1 до n.

3. Написати програму, яка обчислює суму квадратів всіх парних чисел до заданого n.

4. Обчислити значення суми $s=n^2+n^3$, n змінюється від 1 до 5.

5. Обчислити значення суми $s=4n+n(n-1)$, n змінюється від 1 до 50.

6. Обчислити значення функції $z=a\cdot\sin(x)+\cos(ax)^2$ для x від 1 до 25.

7. Обчислити значення функції $z=a\cdot\sqrt{a}\,tg(x)$+a для x від 1 до 30.

8. Обчислити значення виразу 54n+6n до заданого n.

9. Обчислити значення функції z =45(n+1), де n змінюється від 1 до 8.

10. Обчислити значення суми непарних чисел в інтервалі (1, N).

11. Знайти добуток непарних чисел, які знаходяться в інтервалі (1, N).

12. Обчислити значення E=k+ln(N), де k-сума 1/i, i=1,2,3,...N.

13. Знайти суму парних чисел, які знаходяться в інтервалі (1, N).

14. Обчислити значення функції $y=\cos x+\cos x^2+...+\cos x^{30}$ ...

15. Обчислити значення функції y=1!+2!+...+ n!.

16. Обчислити добуток (1-1/2)(1-1/3)...(1-1/n), де n> 2.

17. Обчислити суму степеня числа 2 від 1 до n.

18. Обчислити k кількість цифр в десяткового запису цілого позитивного числа n.

19. Обчислити значення функції y=sin1+sin1.1+sin1.2+sin1.3+...+sin2.

20. Обчислити добуток (1+1/1)(1+1/2)...(1+2/n).

21. Знайти добуток парних чисел від 1 до n, які не входять до інтервалів (10..20) і (30..40)

22. Обчислити y=1+1/2!+1/3!+...+1/n!.

23. Обчислити суму доданків 1/(i-5), i змінюється від 1 до n.

24. Обчислити y=(1!-1)+(2!-2)+...+(n!- n).

25. Обчислити добуток $y=(1-1/(2-1))(1-1/(3-1))...(1-1/(n-1))$, де $n>2$.

26. Обчислити значення функції $y=tg1+tg1.1+tg1.2+tg1.3+...+tg2$.

27. Дано натуральне число. Знайти суму його останніх n цифр.

28. Дано п'ятизначне число. Знайти число, що отримується при прочитанні його цифр справа наліво.

29. Дано натуральні числа x, y. Обчислити добуток x і y, використовуючи лише операцію додавання.

30. Знайти суму $-1^2+2^2-3^2+4^2-5^2+6^2-7^2+8^2-9^2+10^2$. Умовний оператор не використовувати.

2.7 Методичні вказівки

1. Розробити неформальний (словесний) алгоритм розв’язання задачі.
2. Побудувати відповідний математичний алгоритм.
3. Створити блок-схему.
4. Написати програму мовою програмування високого рівня С++.

3 Робота з масивами і вказівниками

Мета: Отримання навичок обробки статичних і динамічних масивів.

3.1 Короткі теоретичні відомості

3.1.1 Визначення масива

Має синтаксис:

```
тип імя [кількість_елементів]={ініціалізітор};
```

Визначення масиву містить тип елементів, ім'я масиву і кількість елементів в масиві.

```
int mas[10];
```

Масиви індексуються, починаючи з 0, тому границі не вказуються, а задаються за замовчуванням. Масив `mas` схематично можна представити так (рис.3.1):

0	1	2	3	4	5	6	7	8	9

Рисунок 3.1 - Схематичне подання масиву

Індекси елементів в масиві `mas` можуть змінюватися від 0 до 9, всього в масиві 10 елементів.

3.1.2 Ініціалізація масива

Ініціалізація масивів можлива при їх визначенні:

```
double d[] = {1, 2, 3, 4, 5};
```

при цьому розмір масиву можна не вказувати, він обчислюється компілятором за кількістю значень перерахованих в фігурних дужках.

3.2 Приклад роботи з одновимірними масивами

Написати програму, яка для цілочисельного масиву зі 100 елементів визначає, скільки позитивних елементів розташовується між його максимальним і мінімальним елементами.

Запишемо алгоритм в найзагальнішому вигляді:

1. Визначити, де в масиві розташовані його максимальний і мінімальний елементи, тобто знайти їх індекси.

2. Переглянути всі елементи, розташовані між ними. Якщо елемент масиву більше нуля, збільшити лічильник елементів на одиницю.

Перед написанням програми корисно скласти тестовий приклад, щоб більш наочно уявити собі алгоритм вирішення задачі. Нижче представлений приклад масиву з 10 чисел і позначені шукані величини (рис.3.2):

6	-8	17	10	-2	4	7	-10	11	8
		max	+		+	+	min		

Рисунок 3.2 - Приклад масиву для пошуку кількості позитивних елементів між максимумом та мінімумом

Порядок розташування елементів у масиві заздалегідь невідомий, і спочатку може слідувати як максимальний, так і мінімальний елемент, крім того, вони можуть і збігатися. Тому перш ніж переглядати масив в пошуках кількості позитивних елементів, потрібно визначити, який з цих індексів більше. Запишемо уточнений алгоритм:

1. Визначити, де в масиві розташовані його максимальний і мінімальний елементи:

• задати початкові значення для індексів максимального і мінімального елементів (наприклад, рівні нулю, але можна використовувати будь-які інші значення індексу, що не виходять за межі масиву);

• переглянути масив, по черзі порівнюючи кожен його елемент з раніше знайденими максимумом і мінімумом. Якщо черговий елемент більше раніше знайденого максимуму, прийняти цей елемент за новий максимум (тобто запам'ятати його індекс). Якщо черговий елемент менше раніше знайденого мінімуму, прийняти цей елемент за новий мінімум.

2. Визначити межі перегляду масиву для пошуку позитивних елементів, що знаходяться між його максимальним і мінімальним елементами:

• якщо максимум розташований в масиві раніше мінімума, прийняти ліву межу перегляду рівній індексу максимуму, інакше - індексу мінімуму;

• якщо максимум розташований в масиві раніше мінімума, прийняти праву межу перегляду рівній індексу мінімуму, інакше - індексу максимуму.

3. Обнулити лічильник кількості позитивних елементів. Переглянути масив в зазначеному діапазоні. Якщо черговий елемент більше нуля, збільшити лічильник на одиницю.

Для економії часу при налагодженні значення елементів масиву задаються шляхом ініціалізації.

```
// Лістинг 3.1. Пошук кількості позитивних елементів
// між максимумом та мінімумом
int main( )
{ const int n=10;
  int a[n]={6, -8, 17, 10, -2, 4, 7, -10, 11, 8};
  int i, imax, imin, count;
  for (i=imax=imin=0; i < n; i++)
  { if (a[i]>a[imax]) imax=i; if(a[i]<a[imin]) imin=i;  }
```

```
    cout << "\n\t max=" << a[imax] << "min=" << a[imin];
    int ibeg=imax<imin ? imax : imin;
    int iend=imax<imin ? imin : imax;
    cout << "\n\t ibeg=" << ibeg << " iend=" << iend;
    for (count=0, i=ibeg+1; i<iend; i++)
      if (a[i]> 0) count++;
    cout << "Кількість позитивних: " << count << endl;
}
```

В програмі використана управляюча послідовність `\t`, яка задає відступ при виведенні на наступну позицію табуляції.

Після знаходження кожної величини доданий налагоджувальний друк. Рекомендується ніколи не нехтувати цим способом налагодження і не шкодувати часу, намагаючись зробити цей друк таким, що добре читається, тобто містить необхідні пояснення і добре відформатований. Це економить багато часу при налагодженні програми.

Масив проглядається, починаючи з елемента, наступного за максимальним (мінімальним), до елемента, що передує мінімальному (максимальному). Індекси меж перегляду зберігаються в змінних `ibeg` і `iend`.

Можна зробити й по-іншому: переглядати масив завжди від максимуму до мінімуму, а індекс при перегляді збільшувати або зменшувати в залежності від їх взаємного розташування.

У наведеній вище програмі для визначення їх значень використовується тернарна умовна операція.

```
int ibeg=imax < imin ? imax : imin;
```

Якщо `imax` менше `imin`, тоді `ibeg` набуває значення `imax` і `imin` в протилежному випадку. Ця конструкція працює швидше ніж умовний оператор

```
int ibeg;
if (imax<imin) ibeg=imax;
               else ibeg=imin;
```

3.3 Вказівники

Кожна змінна в програмі це об'єкт, який має ім'я і значення, за яким можна звернутися до змінної і отримати її значення. Вираз `&a` дозволяє отримати адресу ділянки пам'яті, виділеної змінній `a`. Операція `&` (взяття адреси) застосовна тільки до об'єктів, що мають ім'я і розміщені в пам'яті. Маючи можливість визначити адресу змінної за допомогою операції `&`, треба мати можливість працювати з цією адресою: зберігати її, передавати, перетворювати. Для цього вводиться поняття вказівника. *Вказівник* - це змінна, значенням якої служить адреса об'єкта конкретного типу. Нульова адреса позначається константою `NULL`, яка визначена в заголовному файлі `stdio.h`. Щоб визначити вказівник треба повідомити на об'єкт якого типу посилається цей вказівник.

```
char *z;
int *k,*i;
```

```
float *f;
```

`*` в даному контексті вказує, що змінна після `*` - це вказівник і в якості свого значення зберігає адресу.

Подібно будь-яким змінним змінна типу вказівник має ім'я, адресу в пам'яті і значення (адреса об'єкта) (рис.3.3).

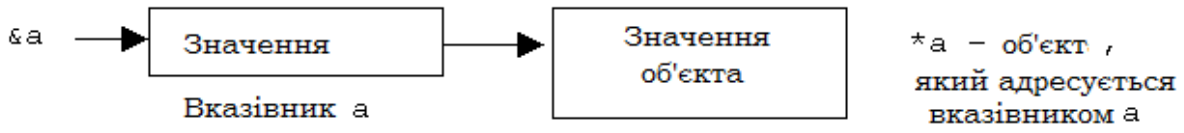

Рисунок 3.3 – Змінна типу вказівник

Операції над вказівниками:

- присвоєння/ініціалізація (=);
- отримання значення об'єкта, на який посилається вказівник (*) - розіменування;
- отримання адреси самого вказівника (&);
- арифметичні операції (інкремент, декремент, складання/віднімання з числом, віднімання вказівників);
- операції порівняння.

Присвоєння/ініціалізація. Використання неініціалізованих вказівників - одна з найпоширеніших помилок. Вказівнику можна присвоїти або адресу об'єкта того ж типу, або значення іншого вказівника, або значення константи `NULL` наприклад:

```
int date = 10;
int *i, *k, *z;
i = &date;
k = i;
z = NULL;
```

У цьому фрагменті наведений приклад ініціалізації вказівників. Вказівник `i` ініціалізований адресою змінної `date` за допомогою операції взяття адреси `&`, вказівник `k` ініціалізований значенням вказівника `i` за допомогою операції присвоєння (вони будуть вказувати на один і той же об'єкт `date`), вказівник `z` ініціалізований значенням `NULL` (нульовий вказівник (`null pointer`) - це вказівник , який не вказує ні на який об'єкт, в C/C++ можна замість `NULL` писати `0`). Вказівник можно ініціалізувати відразу при описі:

```
int date = 10;
int *i=&date, *k= i, *z= NULL;
```

Отримання адреси вказівника. Адресу вказівника можна отримати за допомогою операції взяття адреси `&`, наприклад `&i`.

Операція розйменування (`*`). Операндом цієї операції завжди є вказівник. Результат операції - це той об'єкт, який адресує `вказівник_операнд`, наприклад:

```
int e, c, b, *m;
m = &e ;
*m = c + b ;
```

У цьому фрагменті вказівник `m` ініціалізований адресою змінної `e`, далі за допомогою операції розйменування значення об'єкта, що адресується (змінної `e`), стає рівним сумі змінних `c` і `b`. Варто зауважити, що даний приклад носить ілюстративний характер і набагато простіше написати `e=c+b;`.

Маленьке зауваження: *при описі вказівника `*` вказує, що це вказівник, якщо ж вказівник вже описаний, то `*` перед ним говорить про операцію розйменування.*

Арифметичні операції. За допомогою унарних операцій `++` і `--`, а також при додаванні та відніманні числа, числові значення змінних типу вказівник змінюються по різному, в залежності від типу даних, з яким пов'язані ці змінні, наприклад:

```
char *z;
int *k,*i;
float *f;
z++; //значення змінюється на 1
i++; //значення змінюється на 4
f++; //значення змінюється на 4
```

Тобто при зміні вказівника на 1, вказівник переходить до початку наступного (попереднього) поля тієї довжини, яка визначається типом об'єкта, що адресується вказівником. У бінарних операціях додавання і віднімання можуть брати участь вказівник і величина типу `int`. При цьому результатом операції буде вказівник на вихідний тип, а його значення буде на вказане число елементів більше або менше вихідного. Арифметичні операції з вказівниками мають сенс тільки при роботі з послідовностями однотипних даних (масивами), в інших випадках виконання арифметичних операцій над вказівниками може привести до помилок. Віднімання одного вказівника з іншого в більшості випадків має сенс тільки тоді, коли обидва вказівники вказують на один об'єкт, - як правило, масив. В результаті віднімання отримаємо кількість елементів базового типу, що знаходяться між вказівниками. *Складати вказівники не можна*!

До вказівників можуть застосовуватися операції порівняння `>`,`> =`, `<`, `<=`, `==`,`! =`. Операції порівняння застосовуються тільки до вказівників одного типу і до значень `NULL` і `nullptr`, наприклад:

```
int a = 10;
int b = 20;
int *pa = &a;
int *pb = &b;
if(pa > pb)
   cout<<"pa("<<pa<<") is greater than pb("<<pb<<")";
 else cout<<"pa("<<pa<<") is less or equal pb("<<pb<<")";
```

3.4 Масиви та вказівники

У С ++ вказівники і масиви тісно пов'язані. За допомогою вказівників можна маніпулювати елементами масиву, як і за допомогою індексів.

Ім'я масиву по суті є адресою його першого елемента (`a=&a[0];`). Відповідно через операцію розйменування ми можемо отримати значення за цією адресою:

```
int a[] = {1, 2, 3, 4, 5};
cout << "адреса a[0]= " << a << endl;     //адреса початку масива
cout << "a[0]= " << *a << endl;     // a[0]=1
```

Додаючи до адреси першого елемента деяке число, ми можемо отримати певний елемент масива. Наприклад, в циклі пробіжимося по всіх елементах:

```
for(int i=0; i < 4; i++)
 {cout<<"a["<<i<<"]:address="<<a+i<<"\tvalue="<<*(a+i)<<endl; }
```

Але при цьому ім'я масиву це *константний вказівник*, і на відміну від звичайного вказівника, ми не можемо змінити його адресу, наприклад, так:

```
int a[5] = {1, 2, 3, 4, 5};
a++;              // так не можна
int b = 8;
a = &b;           // так також не можна
```

3.5 Динамічні одновимірні масиви

Якщо до початку роботи програми невідомо, скільки в масиві елементів, в програмі слід використовувати динамічні масиви. Пам'ять під них виділяється за допомогою операції `new` або функції `malloc` в динамічній області пам'яті (купі) під час виконання програми. Адреса початку масиву при цьому зберігається у вказівнику, наприклад:

```
int n=10;                                   //1
int *a=new int[n];                          //2
double *b=(double *)malloc(n*sizeof(double)); //3
```

У першому рядку описується ціла зміна `n`, яка ініціалізується числом 10. Вказівнику на цілу величину `a` присвоюється адреса початку безперервної області динамічної пам'яті, виділеної за допомогою операції `new` (другий рядок). Виділяється стільки пам'яті, скільки необхідно для зберігання `n` величин типу `int`. Величина `n` може бути змінною.

У третьому рядку для виділення пам'яті під `n` елементів типу `double` використовується функція `malloc`, успадкована з бібліотеки С, яка виділяє місце в пам'яті для `n` елементів типу `double`. В даному операторі виконано необхідно приведення типу до вказівника типу `double`.

Обнулення пам'яті при її виділенні не відбувається. Ініціалізувати динамічний масив не можна.

Звернення до елемента динамічного масива здійснюється так само, як і до елемента звичайного - наприклад `a[3]`. Можна звернутися до елемента масиву і іншим способом - `*(a+3)`. В цьому випадку ми явно задаємо ті ж дії, що виконуються при зверненні до елемента масива звичайним чином. Розглянемо їх докладніше. В змінній-вказівнику `a` зберігається адреса початка масиву (тобто нульового елемента `a[0]`). Для отримання адреси третього елемента до цієї адреси додається зміщення 3. Операція додавання з константою для вказівників враховує розмір адресованих елементів, тобто насправді індекс множиться на довжину елемента масива: `a+3*sizeof(int)`. Потім за допомогою операції `*` (розйменування) виконується вибірка значення зі зазначеної ділянки пам'яті.

Після закінчення роботи з динамічним масивом необхідно звільнити пам'ять за допомогою операції `delete[]`, наприклад:

```
delete [] a;
```

Розмірність масиву при цьому не вказується.

! Квадратні дужки в операції `delete[]` при звільненні пам'яті з-під масиву обов'язкові. Їх відсутність може привести до того, що буде видалений тільки перший елемент масиву, а решта стануть недоступні і будуть займати місце в пам'яті. Пам'ять, виділену за допомогою `malloc`, слід звільняти за допомогою функції `free`.

```
free(a);
```

Таким чином, час життя динамічного масиву, як і будь-якої динамічної змінної, - з моменту виділення пам'яті до моменту її звільнення. Область дії залежить від місця опису вказівника, через який проводиться робота з масивом. Область дії і час життя вказівників підкоряються загальним правилам, локальна змінна при виході з блоку, в якому вона описана, «губиться». Якщо ця змінна є вказівником і в ній зберігається адреса виділеної динамічної пам'яті, при виході з блоку ця пам'ять перестає бути доступною, проте не позначається як вільна, тому не може бути використана в подальшому. Це називається *витоком пам'яті* і є поширеною помилкою:

```
{ int n; // приклад витоку пам'яті
  cin >> n;
  int *pmas = new int[n];
} // після виходу з блоку вказівник pmas недоступний,
  // і пам'ять зайнята
```

Приклад. *Написати програму, яка для дійсного масиву з n елементів визначає суму його елементів, розташованих правіше останнього негативного елементу.*

У цьому завданні розмірність масиву задана змінною величиною. Передбачається, що вона буде нам відома на етапі виконання програми до того, як ми будемо вводити самі елементи. В цьому випадку ми зможемо виділити в динамічній пам'яті безперервну ділянку потрібного розміру, а

потім заповнювати її значеннями, що вводяться. Якщо ж стоїть завдання вводити заздалегідь невідому кількість чисел до тих пір, поки не буде введено будь-яку ознаку закінчення введення, то заздалегідь виділити достатню кількість пам'яті не вдасться і доведеться скористатися так званими динамічними структурами даних, наприклад списком. Поки зупинимося на першому припущенні - що кількість елементів масиву вводиться із клавіатури до початку введення самих елементів.

Можливий алгоритм рішення цієї задачі: переглядаючи масив з початку до кінця, знайти номер останнього негативного елементу, а потім організувати цикл підсумовування всіх елементів, розташованих правіше нього. Ось як виглядає побудована за цим алгоритмом програма:

```
// Лістинг 3.2. Робота з динамічним масивом. Знаходження
// суми елементів після останнього негативного елементу
#include <iostream>
int main()
{  int n;
   cout << " Введіть кількість елементів ";
   cin >> n;
   int i, ineg;
   float sum, *a=new float [n];                          //1
   cout << " Введіть елементи масива ";
   for (i=0; i < n; i++) cin >> a[i];
   for (i=0; i < n; i++) cout << a[i] << ' ';            //2
   for (i=0; i < n; i++)
     if (a[i] < 0) ineg=i;                               //3
   for (sum=0, i=ineg + 1; i < n; i++)
     sum += a[i];                                       //4
   cout << " Сума= " << sum;
   delete [] a;
}
```

Оскільки кількість елементів заздалегідь не задана, пам'ять під масив виділяється в рядку 1 на етапі виконання програми за допомогою операції `new`. Виділяється стільки пам'яті, скільки необхідно для зберігання `n` елементів дійсного типу, і адреса початку цієї ділянки заноситься до вказівника `a`. Потім в циклі вводиться поелементно весь масив.

Номер останнього негативного елементу масиву зберігається в змінній `ineg`. При перегляді масиву в циклі за допомогою оператора `if` (рядок 3) в цю змінну послідовно записуються номери всіх негативних елементів масиву, таким чином, після виходу з циклу в ній залишається номер самого останнього.

З метою оптимізації програми може виникнути думка об'єднати цикл знаходження цього номера з циклами введення і контрольного виводу елементів масиву, але не варто так робити, тому що введення даних, їх виведення і аналіз - різні за змістом дії, і змішування їх в одному циклі не додасть програмі ясності. Після налагодження програми контрольне виведення (рядок 2) можна видалити або закоментувати.

Для масивів, що містять негативні елементи, ця програма працює вірно, але при їх відсутності, як правило, завершується аварійно. Це пов'язано з тим, що якщо в масиві немає жодного негативного елементу, змінній `ineg` значення не присвоюють. Тому в операторі `for` (рядок 4) буде використано невизначене значення `ineg`. Тому в програму необхідно внести перевірку, чи є в масиві хоча б один негативний елемент. Для цього змінній `ineg` присвоюється початкове значення, що не входить до множини допустимих індексів масиву (наприклад, -1). Після циклу пошуку номера негативного елементу виконується перевірка, чи зберіглося початкове значення `ineg` незмінним. Якщо це так, це означає, що умова `a[i]<0` в рядку 3 не виконалася жодного разу, і негативних елементів в масиві немає:

```
// Лістинг 3.3. Модифікований лістинг 3.2.
#include <iostream.h>
int main()
{ int n;
  cout << " Введіть кількість елементів ";
  cin >> n;
  int i;
  float sum, *a=new float [n];                          //1
  cout << "Введіть елементи масива ";
  for (i=0; i < n; i++) cin >> a[i];
  for (i=0; i < n; i++) cout << a[i] << ' ';            //2
  int ineg=-1;
  for (i=0; i < n; i++) if (a[i] < 0) ineg=i;           //3
     if (ineg != -1)
          { for (sum=0, i=ineg + 1; i < n; i++)
               sum += a[i];   //4
            cout << "\n Сума " << sum << endl;
          }
          else cout<<"\nНегативних елементів немає"<<endl;
  delete [] a;
}
```

Якщо не зупинятися на досягнутому, можна запропонувати і більш раціональне рішення цього завдання: переглядати масив в зворотному порядку, підсумовуючи його елементи, і завершити цикл, як тільки зустрінеться негативний елемент.

3.6 Двовимірні масиви

Відповідно до синтаксису в С ++ існують тільки одновимірні масиви, але їх елементами, в свою чергу, теж можуть бути масиви.

```
int a[5][5];
```

При визначенні масива за допомогою операторів опису, обидві його розмірності повинні бути константами або константними виразами, оскільки інструкції по виділенню пам'яті формуються компілятором до виконання програми. наприклад:

```
int a[3][5]; // Цілочисельна матриця з 3 рядків і 5 стовпців
```

Масив зберігається по рядках в безперервній області пам'яті. Рядки масиву нічим не відокремлені один від одного, тобто прямокутною матрицею двовимірний масив є тільки в нашій уяві. У пам'яті спочатку розташовується одновимірний масив `a[0]`, що представляє собою перший рядок масива `a` (але з індексом 0), потім - масив `a[1]`, що представляє собою другий рядок масива `a`, і т.д. Кількість елементів в кожному з цих масивів дорівнює довжині рядка, тобто кількості стовпців в матриці. При перегляді масиву від початку в першу чергу змінюється правий індекс (номер стовпчика).

Для доступу до окремого елемента масива застосовується конструкція виду `a[i][j]`, де `i` (номер рядка) і `j` (номер стовпчика) - вирази цілочисельного типу. *Кожен індекс може змінюватися від 0 до значення відповідної розмірності, зменшеної на одиницю.*

!Перший індекс завжди сприймається як номер рядка, другий - як номер стовпчика, незалежно від імені змінної.

Можна звернутися до елемента масива і іншими способами: `*(*(a+i)+j)` або `*(a[i]+j)`. Вони наведені для кращого розуміння механізму індексації, оскільки тут в явному вигляді записані ті ж дії, які генеруються компілятором при звичайному зверненні до масиву. Розглянемо їх докладніше.

Припустимо, потрібно звернутися до елемента, розташованого на перетині другого рядка і третього стовпчика - `a[2][3]`. Як і для одновимірних масивів, ім'я масиву `a` являє собою константний вказівник на початок масиву. В даному випадку це масив, що складається з трьох масивів (рядків). Спочатку потрібно звернутися до другого рядку масива, тобто одновимірного масиву `a[2]`. Для цього треба додати до адреси початку масиву зміщення, рівне номеру рядка, і виконати розадресацію (розйменування): `*(a+2)`. При додаванні вказівника з константою враховується довжина елемента, який адресується, тому насправді додається число 2, помножене на довжину елемента, тобто `2*(5*sizeof(int))`, оскільки елементом є рядок, що складається з 5 елементів типу `int`.

Далі потрібно звернутися до третього елементу отриманого масиву. Для отримання його адреси знову застосовується додавання вказівника з константою 3 (насправді додається `3*sizeof(int)`), а потім застосовується операція розйменування для отримання значення елемента: `*(*(a+2)+3)`.

У випадку використання нотації `*(a[i]+j)` за індексом `i` ми звертаємося до першого елемента потрібного рядка, а потім додаємо до цієї адреси зміщення, яке дорівнює `j`, щоб дістатися до потрібного нам елемента рядка і розйменовуємо цю адресу `*(a[i]+j)`. Для елемента `a[2][3]` така нотація буде виглядати `*(a[2]+3)`.

При описі масива можна задати початкові значення його елементів. Їх записують у фігурних дужках. Елементи масиву ініціалізуються в порядку їх розташування в пам'яті. Наприклад, оператор

```
int a[3][5]={1, 2, 1, 3, 5, 2, 3, 4, 5, 1, 1, 3, 2, 6, 1 };
```

визначає матрицю з наступними значеннями елементів:

```
1 2 1 3 5
2 3 4 5 1
1 3 2 6 1
```

Якщо кількість значень у фігурних дужках перевищує кількість елементів в масиві, при компіляції буде видано повідомлення про помилку. Якщо значень менше, елементи масиву, що залишилися, *ініціалізуються значенням за замовчуванням* (для основних типів це 0). Можна задавати початкові значення не для всіх елементів масива. Для цього список значень констант для кожного рядка укладається в додаткові фігурні дужки. Ось, наприклад, як заповнити одиницями нульовий і перший стовпці наведеного вище масива:

```
int a[3][5]={{1, 1}, {1, 1}, {1, 1}};
```

Інші елементи масива обнуляються.

При явному вказанні хоча б одного ініціалізуючого значення для кожного рядка кількість рядків масиву можна не вказувати; пам'ять буде виділена під стільки рядків, скільки серій значень у фігурних дужках вказано в списку, наприклад:

```
int a[][5]={{1, 1, 7, 7, 7}, {1, 1, 0}, {1, 1, 2, 2, 2}};
```

в цьому випадку у масиві `a` буде 3 рядки.

Приклад. Знаходження середнього арифметичного і кількості позитивних елементів**.**

Написати програму, яка для цілочисельної матриці 10x20 визначає середнє арифметичне її елементів і кількість позитивних елементів в кожному рядку.

Алгоритм рішення цієї задачі: для обчислення середнього арифметичного елементів масиву потрібно знайти їх загальну суму, після чого розділити її на кількість елементів. Порядок перегляду масиву (по рядках або по стовпцях) ролі не грає. Визначення кількості позитивних елементів кожного рядка вимагає перегляду матриці по рядках. Обидві величини обчислюються за один перегляд матриці.

```
// Лістинг 3.4. Знаходження середнього арифметичного
// і кількості позитивних елементів двовимірного масиву
#include <iostream.h>
#include <iomanip.h>
void main()
{ const int nrow = 5, ncol = 10;
  int a[nrow][ncol];
  int i, j;
  cout << "Введіть елементи масива:" << endl;
  for (i = 0; i < nrow; i++)
```

```
      for (j = 0; j < ncol; j++)
           cin >> a[i][j];
   for (i = 0; i < nrow; i++)
   {  for (j = 0; j < ncol; j++)
           cout << setw(4) << a[i][j] << ' ';
      cout << endl;
   }
   int n_pos_el;
   float s = 0;
   for (i = 0; i < nrow; i++)
   { n_pos_el = 0;
     for (j = 0; j < ncol; j++)
      { s+=a[i][j];
        if (a[i][j] > 0) n_pos_el++;
      }
     cout<<Рядок:"<<i<<" кількість:"<<n_pos_el<< endl;
   }
   s/=nrow*ncol;
   cout << "Середнє арифметичне: " << s << endl;
}
```

Розмірності масиву задані іменованими константами, що дозволяє легко їх змінювати. При обчисленні кількості позитивних елементів для кожного рядка виконуються однотипні дії: обнулення лічильника, перегляд кожного елемента рядка і порівняння його з нулем, у випадку співпадання - збільшення лічильника на одиницю, а після закінчення обчислень - виведення результуючого значення лічильника. Слід звернути увагу на два моменти. По-перше, потрібно ще до написання алгоритму вирішити, яким чином будуть зберігатися результати. Кількість позитивних елементів для кожного рядка своя, і в результаті повинні отримати стільки значень, скільки рядків в матриці. В цьому завданні можна відвести для зберігання цих значень одну єдину змінну цілого типу, оскільки вони обчислюються послідовно, після чого виводяться на екран. Однак в інших завданнях ці значення можуть згодом знадобитися одночасно. В цьому випадку для їх зберігання доведеться описати цілочисельний масив з кількістю елементів, що дорівнює кількості рядків матриці.

Другий важливий момент - місце ініціалізації суми і кількості. Сума обнуляється перед циклом перегляду всієї матриці, а кількість позитивних елементів - перед циклом перегляду чергового рядка, оскільки для кожного рядка його обчислення починається заново.

!*Записуйте оператори ініціалізації величин, які накопичуються в циклі, безпосередньо перед циклом, в якому вони обчислюються.*

Після введення значень передбачено їх контрольне виведення на екран. Для того щоб елементи матриці розташовувалися один під іншим, використовується маніпулятор `setw(4)`, що встановлює для чергового виведеного значення ширину поля в чотири символи. Для використання маніпулятора необхідно підключити до програми заголовний файл `<iomanip.h>`. Після кожного рядка виводиться символ переводу рядка `endl`.

При написанні вкладених циклів стежте за відступами. Всі оператори одного рівня вкладеності повинні починатися в одній і тій же колонці. Це полегшує читання програми і, отже, пошук помилок.

3.6.1 Динамічні двовимірні масиви

У динамічній області пам'яті (купі) можна створювати двовимірні масиви за допомогою операції `new` або функції `malloc`. Зупинимося на першому варіанті, оскільки він більш безпечний і простий у використанні.

При виділенні пам'яті відразу під весь масив кількість рядків (найлівішу розмірність) можна задавати за допомогою змінної або виразу, а кількість стовпців має бути константним виразом, тобто явно визначеним до виконання програми. Після слова `new` записується тип створюваного масиву, а потім - його розмірності в квадратних дужках (аналогічно опису «звичайних», нединамічних масивів), наприклад:

```
int n;
const int m=5;
cin >> n;
int(*a)[m]=new int [n][m];                          //1
int **b=(int **) new int [n][m];                    //2
```

В цьому фрагменті показано два способи створення динамічного двовимірного масиву. В операторі 1 адреса початку виділеної за допомогою `new` ділянки пам'яті присвоюється змінній `a`, визначеній як вказівник на масив з `m` елементів типу `int`. Саме такий тип значення повертає в даному випадку операція `new`. Дужки () необхідні, оскільки без них конструкція інтерпретувалася б як масив вказівників. Всього виділяється пам’ять під `n` елементів.

В операторі 2 адреса початку виділеної ділянки пам'яті присвоюється змінній `b`, яка описана як «`вказівник на вказівник на int`», тому перед присвоєнням потрібно перетворення типу `(int **)`.

Недолік подібного виділення пам'яті полягає в тому, що змінною може здаватися тільки крайня ліва розмірність масиву, а решта розмірності повинні бути константами. Такого недоліку позбавлений наступний спосіб виділення пам'яті - більш універсальний і безпечний спосіб виділення пам'яті під двовимірний масив, коли обидві його розмірності задаються на етапі виконання програми, наведений в лістингу 3.5:

```
// Лістинг 3.5. Динамічне виділення пам'яті під двовимірний
// масив
int nrow, ncol;
cout << "Введіть кількість рядків і стовпців:";
cin >> nrow >> ncol;
int **a = new int *[nrow];                          //1
for (int i = 0; i < nrow; i++)                      //2
     a[i] = new int [ncol];                         //3
```

В операторі 1 оголошується змінна типу «вказівник на вказівник на int» і виділяється пам'ять під масив вказівників на рядки масиву (кількість рядків - nrow). В операторі 2 організовується цикл для виділення пам'яті під кожен рядок масиву. В операторі 3 кожному елементу масиву вказівників на рядки присвоюється адреса початку ділянки пам'яті, виділеної під рядок двовимірного масиву. Кожен рядок складається з ncol елементів типу int.

На відміну від звичайного двовимірного масиву для динамічного масиву виділена пам'ять не є суцільною ділянкою пам'яті, так як вона виділяється за допомогою декількох операцій new (рис.3.4) :

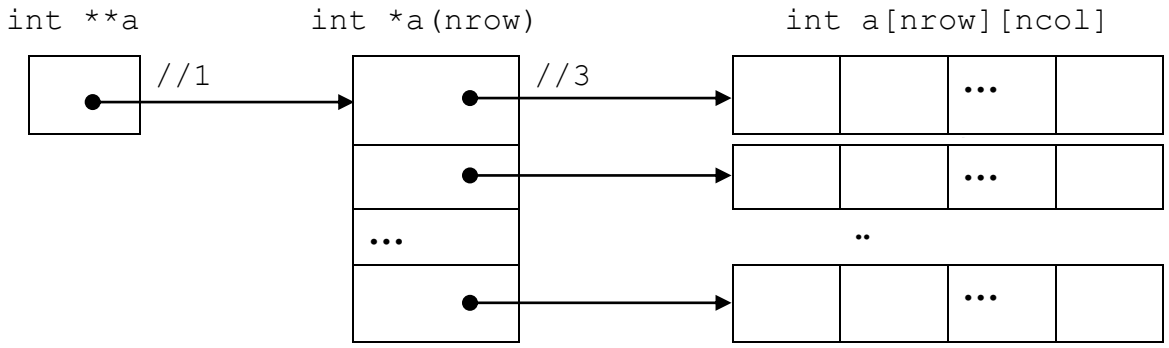

Рисунок 3.4 - Виділення пам’яті під двовимірний динамічний масив

Спочатку виділена пам’ять під одновимірний масив (вектор-стовпець) вказівників *a, адреса цієї ділянки зберігається у вказівнику **a. Далі в циклі кожному елементу цього вектор-стовпця присвоюється адреса рядка динамічного двовимірного масива a[i]= new int[ncol];. Ці рядки можуть розташовуватися в купі де завгодно.

Звернення до елементів динамічних масивів проводиться точно так же, як до елементів «звичайних», за допомогою конструкції виду a[i][j].

Після закінчення роботи з динамічним двовимірним масивом необхідно звільнити пам'ять. Робимо це в зворотному виділенню пам'яті порядку:

```
for (int i = 0; i < nrow; i++)
     delete [] a[i];     // видаляємо елементи масиву
delete [] a;              //видаляємо масив вказівників
```

Розмірність масиву при цьому не вказується. ***Необхідно обов’язково вказувати*** ***[]***, тому що їх відсутність компілятор не рахує за помилку, вцьому випадку він видалить тільки перший елемент відповідного масиву, а інші будуть захаращувати купу. Контроль наявності [] повністю лежить на програмістові.

3.7 Функції для роботи з випадковими числами в С++

Для ефективного використання в програмі випадкових чисел необхідні два засоби:

1. *функція-генератор*, яка при кожному виклику буде повертати випадкове число;

2. *функція-ініціалізатор* для випадкової ініціалізації генератора.

Необхідність в другій функції викликана тим, що випадкові числа, отримані програмним шляхом, насправді є псевдовипадковими. Вони обчислюються за певною формулою, і при однакових початкових умовах послідовність чисел буде повторюватися. При виконанні функції-ініціалізатора для функції-генератора встановлюється нове початкове значення.

Випадкові числа в C ++ можуть бути згенеровані функцією `rand()` зі стандартної бібліотеки C ++. Сумісність: *POSIX, Win32, ANSI C, ANSI C ++*.

Прототип: `int rand (void);`

мультиплікативний конгруентний генератор випадкових чисел з періодом 2 в 32 степені. При кожному виклику повертає наступне псевдовипадкове число в межах від 0 до `RAND_MAX`. `RAND_MAX` - символьна константа, оголошена в файлі `<stdlib.h>`, яка визначає найбільше випадкове число. Для MVS гарантується мінімальне значення `RAND_MAX = 32767`, але воно може бути і більше, в залежності від компілятора.

```
/* (в модуле stdlib.h) */ #define RAND_MAX 32767;
//Лістинг 3.6. Приклад роботи функції rand
#include <stdio.h>
#include <stdlib.h>
int main(void)
{  int i;
   printf("10 випадкових чисел від 0 до 99 \n");
   for (i=0; i<10; i++) printf("%d\n", rand()%100);
}
```

Якщо ми запустимо цю програму кілька разів, ми отримаємо одні й ті ж числа при генерації. Справа в тому, що функція `rand`() один раз генерує випадкові числа, а при наступних запусках програми всього лише відображає згенеровані перший раз числа. Така особливість функції `rand()` потрібна для того, щоб можна було правильно налагодити розроблювану програму. При налагодженні програми, внесши якісь зміни, необхідно упевнитися, що програма спрацьовує правильно, а це можливо, якщо вхідні дані (згенеровані числа) залишилися ті ж. Коли програма успішно налагоджена, потрібно, щоб при кожному виконанні програми генерувалися випадкові числа. Для цього потрібно скористатися функцією `srand()` зі стандартної бібліотеки C ++.

Функція `srand()`, отримавши цілий позитивний аргумент типу `unsigned` або `unsigned int` (беззнаковое ціле), виконує рандомізацію (ініціалізацію), таким чином, щоб при кожному запуску програми функція `rand()` генерувала різні випадкові числа.

Прототип: `void srand (unsigned seed);`

Сумісність: *POSIX, Win32, ANSI C, ANSI C++*.

Початкова ініціалізація відбувається викликом `srand` з аргументом 1. Щоб кожен раз генерувалися нові випадкові числа необхідно, що б змінювався аргумент у функції `srand`. Нове початкове значення

встановлюють викликом цієї функції з іншим аргументом, що не завжди зручно. Щоб виробляти рандомізацію автоматично, тобто, не змінюючи кожного разу аргумент у функції `srand()` можна скористатися функцією `time()` із заголовного файлу <`time.h`>(`ctime`) із аргументом 0.

```
// Лістинг 3.7. Ініціалізація генератора випадкових чисел
// системним часом…
#include <stdlib.h>
#include <time.h>
int main()
{ int a;
  srand(time(0));
     // ініціалізація генератора псевдовипадкових чисел
  a=rand();    // отримуємо чергове псевдовипадкове число
  cout<<a;     // виведення на екран
}
```

Функція `time` повертає поточний час системи, в якості аргументу вона приймає вказівник на змінну типу `time_t`, якій і буде присвоєний системний час. Прототип функції виглядає так:

```
time_t time(time_t* time);
```

Системний час і повертається функцією, і поміщається в переданий аргумент, але можна передавати нульовий вказівник (тобто 0 або `NULL`). 0 в даному випадку не число, а нульовий вказівник.

Іноді при програмуванні виникає потреба в масштабуванні інтервалу генерації випадкових чисел. Для того щоб масштабувати інтервал генерації чисел потрібно скористатися, операцією знаходження залишку від ділення `%`. Можна скористатися формулою (1):

$$num = a + rand() \ \% \ (b+1), \qquad (1)$$

де `a` та `b` - це межі діапазону генерації.

В лістингу 3.8 наведений приклад генрації значень в діапазоні від 0 до 9, від 1 до 10, від -1 до 1.

```
// Лістинг 3.8. Генерація випадкових чисел
// (з масштабуванням інтервалу і без)
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
int main ()
{ int num;
     // ініціалізація генератора псевдовипадкових чисел
  srand (time(NULL) );
  printf ("RAND_MAX: %d \n",RAND_MAX);
  num = rand();
  printf ("Guess the number: %d \n",num);
  num = rand() % 10;
  printf ("Guess the number (0 to 9): %d \n",num);
```

```
  num = rand() % 10 + 1;
  printf ("Guess the number (1 to 10): %d \n",num);
  num = (rand() % 2)*2 - 1;
  printf ("Guess the number (-1 or 1): %d \n",num);
}
```

Для генерації негативних значень можна скористатися формулою (2):

```
num = rand()%a – b,                                    (2)
```

де `a` та `b` - це довільні числа, їх значення підбираються дослідним шляхом.

Для отримання чисел з плаваючою точкою можна скористатися формулою (3):

```
f = (тип) rand() / RAND_MAX,                           (3)
```

де `тип` - дійсний тип, `RAND_MAX` - максимальне значення, яке може дати функція `rand`().

Для генерації дійсних чисел в певному діапазоні (від `a` до `b`) можна скористатися формулами (4, 5):

```
f=(rand() % ((b - a) * 1000)) / 1000.0;                (4)
```

такий спосіб дає обмежену десяткову точність (3 знаки в даному випадку), але іноді корисний, при цьому `a` і `b` повинні бути цілими, що є недоліком в деяких випадках, або

```
f= a + (b - a) * (double) rand() / (double)RAND_MAX;   (5)
```

подібна формула дає високу точність, і межі діапазону можуть бути будь-якого типу.

Генератор випадкових чисел дуже зручно використовувати для заповнення масивів:

```
int x, ranvals [20];
for (x = 0; x < 20; ranvals[x++] = rand());
```

3.8 Контрольні запитання

1. Що таке масив?
2. Якого типу можуть бути індекси масивів?
3. Що таке розмірність масивів?
4. Який алгоритм пошуку мінімуму або максимуму в одновимірному масиві?
5. Як знайти індекс елемента в одновимірному масиві за його значенням?
6. Як згенерувати елементи масива?
7. Як звернутися до елементу масиву за вказівником?
8. Для чого використовується динамічна пам'ять в програмуванні?
9. Як довго зберігаються дані в динамічної пам'яті?
10. Які можливі варіанти доступу до динамічної пам'яті?
11. Що повертає операція виділення динамічної пам'яті в разі успішного виконання?

12. Що повертає операція виділення динамічної пам'яті, якщо ділянка необхідного розміру не може бути виділена?
13. Чому при завершенні роботи з динамічною пам'яттю її необхідно звільнити?
14. В чому різниця між статичним та динамічним масивами?
15. Які можуть бути наслідки для роботи програми, якщо не звільняти динамічну пам'ять?
16. Як виділяється пам'ять для двовимірних динамічних масивів?
17. Якими способами можна звернутися до елементів двовимірного динамічного масиву?
18. Назвіть порядок звільнення пам'яті, виділеної під двовимірний динамічний масив.

3.9 Задачі

Елементи масиву згенерувати випадковим чином. Вибрати за своїм номером у списку групи завдання на одновимірні і двовимірні масиви. Додаткові бали можна заробити, динамічно виділяючи пам'ять під елементи масиву.

Завдання 1. Одновимірні масиви:

1.В одновимірному масиві, що складається з n дійсних елементів, обчислити:
1) суму негативних елементів масиву;
2) добуток елементів масиву, розташованих між максимальним і мінімальним елементами.

2.В одновимірному масиві, що складається з n дійсних елементів, обчислити:
1) суму позитивних елементів масиву;
2) добуток елементів масиву, розташованих між максимальним за модулем і мінімальним за модулем елементами.

3.В одновимірному масиві, що складається з n цілих елементів, обчислити:
1) добуток елементів масиву з парними номерами;
2) суму елементів масиву, розташованих між першим і останнім нульовими елементами.

4.В одновимірному масиві, що складається з n дійсних елементів, обчислити:
1) суму елементів масиву з непарними номерами;
2) суму елементів масиву, розташованих між першим і останнім негативними елементами.

5. В одновимірному масиві, що складається з n дійсних елементів, обчислити:
1) максимальний елемент масиву;
2) суму елементів масиву, розташованих до останнього позитивного елемента.

6. В одновимірному масиві, що складається з n дійсних елементів, обчислити:
1) мінімальний елемент масиву;
2) суму елементів масиву, розташованих між першим і останнім позитивними елементами.

7. В одновимірному масиві, що складається з n цілих елементів, обчислити:

1) номер максимального елемента масиву;

2) добуток елементів масиву, розташованих між першим і другим нульовими елементами.

8. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) номер мінімального елемента масиву;

2) суму елементів масиву, розташованих між першим і другим негативними елементами.

9. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) максимальний за модулем елемент масиву;

2) суму елементів масиву, розташованих між першим і другим позитивними елементами.

10. В одновимірному масиві, що складається з n цілих елементів, обчислити:

1) мінімальний за модулем елемент масиву;

2) суму модулів елементів масиву, розташованих після першого елемента, рівного нулю.

11. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) номер мінімального за модулем елемента масиву;

2) суму модулів елементів масиву, розташованих після першого негативного елементу.

12. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) номер максимального за модулем елемента масиву;

2) суму елементів масиву, розташованих після першого позитивного елементу.

13. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) кількість елементів масиву, що лежать в діапазоні від A до B;

2) суму елементів масиву, розташованих після максимального елемента.

14. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) кількість елементів масиву, рівних 0;

2) суму елементів масиву, розташованих після мінімального елемента.

15. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) кількість елементів масиву, більших C;

2) добуток елементів масиву, розташованих після максимального за модулем елемента.

16. В одновимірному масиві, що складається з n дійсних елементів, обчислити;

1) кількість негативних елементів масиву;

2) суму модулів елементів масиву, розташованих після мінімального за модулем елемента.

17.В одновимірному масиві, що складається з n цілих елементів, обчислити:

1) кількість позитивних елементів масиву;

2) суму елементів масиву, розташованих після останнього елемента, рівного нулю.

18. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) кількість елементів масиву, менших С;

2) суму цілих частин елементів масиву, розташованих після останнього негативного елементу.

19. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) добуток негативних елементів масиву;

2) суму позитивних елементів масиву, розташованих до максимального елемента.

20. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) добуток позитивних елементів масиву;

2) суму елементів масиву, розташованих до мінімального елемента.

21.В одновимірному масиві, що складається з n дійсних елементів:

1) поміняти місцями мінімальний і максимальний елементи масиву.

2) обчислити кількість елементів, що перевищують його середнє значення більш, ніж на 10%.

22. В одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) кількість елементів, які збігаються з його мінімальним значенням.

2) кількість елементів, рівних середньому арифметичному масиву.

23. В одновимірному масиві, що складається з n цілих елементів, обчислити:

1) кількість елементів, які збігаються з його максимальним значенням.

2) максимальну суму серед сум виду: a [0] + a [n]; a [1] + a [n-1], a [2] + a [n-2], ...

24. В одновимірному масиві, що складається з n дійсних елементів:

1) отримати послідовність: a [0] -a [n]; a [1] -a [n], a [2] -a [n], ..., 0.

2) обчислити кількість ненульових елементів.

25. В одновимірному масиві, що складається з n дійсних елементів:

1) поміняти місцями перший і останній, другий і передостанній ...

2) обчислити кількість невірних рівності серед: a [0] = a [n: 2 + 1]; a [1] = a [n: 2 + 2], a [2] = a [n: 2 + 3], ...

26. В одновимірному масиві, що складається з n дійсних елементів:

1) поміняти місцями елементи з парними і непарними індексами.

2) обчислити суму елементів масиву, розташованих перед першим негативним елементом.

Приклад. Знайти кут між двома векторами. Використати формулу $\cos\alpha = \frac{(\vec{a},\vec{b})}{|\vec{a}|\cdot|\vec{b}|}$; перехід від радіанів до градусів здійснити за формулою $\varphi_0 = \varphi_r \cdot 180\ \pi$

Лістінг 3.9

```
#include <stdio.h>
#include <stdlib.h>
#define _USE_MATH_DEFINES
#include <math.h>
#define N 10 // максимальна розмірність вектора
void vvid(int *a, int n);
int dobutok(int *a, int *b, int n);

int main()
{  int a[N],b[N];
   int n; // реальна довжина вектора
   double alfa;
   system("chcp 1251 & cls");
   printf("Вкажіть розмірність векторів ");
   scanf("%d",&n);
   printf("Введіть координати вектора a "); vvid(a,n);
   printf("Введіть координати вектора b "); vvid(b,n);
   /* обчислення кута */
   alfa=acos(dobutok(a,b,n)/
   sqrt((double)dobutok(a,a,n)*dobutok(b,b,n)))/M_PI*180.;
   printf("Кут між векторами %0.2f°",alfa);
   printf("\n\n");
   system("pause");
   return 0;
}

/* Ввід координат вектора */
void vvid(int *a, int n)
{  int i;
   for (i=0; i<n; i++) scanf("%d",&a[i]);
}
/* Знаходження скалярного добутку двох векторів */
int dobutok(int *a, int *b, int n)
{  int i;
   int s=0;
   for (i=0; i<n; i++)
   s+=a[i]*b[i];
   return s;
}
```

Приклад. Вивести на екран перший мінімальний і перший максимальний елементи масиву. Цикл пошуку виконати з другого елемента, щоб елемент не порівнювати сам із собою**.**

```
Лістінг 3.10
#include <stdio.h>
#include <stdlib.h>
#define N 10
int main()
{ // пошук мінімального й максимального елементів
  int a[N]={2,4,1,8,9,1,3,9,1,9};
  int kmin, kmax;
      // номери мінімального й максимального елементів
  int i; // номер поточного елемнента
  int amin, amax; // мінімальний і максимального елементи
  system("chcp 1251 && cls");
  amin=a[0]; kmin=0; amax=a[0]; kmax=0;
  for (i=1; i<N; i++)
  { // цикл виконується з другого елемента
    if (a[i]<amin) { amin=a[i]; kmin=i; }
    if (a[i]>amax) { amax=a[i]; kmax=i; }
  }
  printf("Мінімальний елемент a[%d]=%d\n",kmin,amin);
  printf("Максимальний елемент a[%d]=%d",kmax,amax);
  printf("\n\n");
  system("pause");
  return 0;
}
```

Завдання 2. Двовимірні масиви

1. Вивести номер стовпця матриці MxN, в якому добуток ненульових елементів мінімальний.
2. Отримати з матриці A (n, n) матрицю B (n-1, n-1) видаленням n-го рядка і k-го стовпця.
3. Вивести номер рядка матриці MxN, у якому всі елементи впорядковані за спаданням.
4. За масивом A (n, m) отримати масив B (n), присвоївши k-му елементу значення 1, якщо всі елементи k-го стовпця матриці A нульові, інакше - значення 0.
5. Визначити кількість елементів дійсної матриці MxN, які більше суми інших елементів свого рядка.
6. Визначити, чи є задана квадратна ціла матриця розміром MxN симетричною відносно головної діагоналі.
7. У квадратній матриці знайти скалярний добуток рядка, в якому знаходиться найбільший елемент матриці, на стовпець з найменшим елементом.
8. Замість співпадаючих чисел в рядках цілої матриці MxN записати нулі.
9. Визначити суму елементів цілої матриці MxN, які одночасно є найбільшими в своєму стовпці і найменшими в своєму рядку.
10. Матриця MxN умовно складається з 4х матриць. Поміняти місцями дві матриці, що стоять на головній діагоналі.

11. Дана дійсна матриця MxN. Знайти матрицю того ж порядку, в якій елемент дорівнює 1, якщо відповідний йому елемент вихідної матриці більше елемента, розташованого в його рядку на головній діагоналі, і дорівнює 0 в іншому випадку.
12. Визначити суму елементів дійсної матриці MxN, розташованих в 2-х вертикальних трикутниках, утворених перетином діагоналей матриці.
13. Є дійсна матриця MxN. Переставити її рядки і стовпці таким чином, щоб найбільший елемент матриці виявився в верхньому лівому кутку.
14. Матриця має парне число рядків. Поміняти місцями 1-ий і останній, 2-ий і передостанній і т.д.
15. Слід квадратної матриці - це сума елементів головної діагоналі. Є матриця порядку m, натуральне число n. Знайти сліди матриць A, A^2, A^3, ..., A^n.
16. Є квадратні матриці A і B. Отримати матрицю AB-BA.
17. Є квадратна матриця A порядку n і вектор b з n елементами. Отримати вектор Ab.
18. Є квадратна матриця A порядку n і вектор b з n елементами. Отримати вектор A^2b.
19. Є квадратна матриця A порядку n і вектор b з n елементами. Отримати вектор (A-E) b, де E-одинична матриця порядку n.
20. Є квадратна матриця A порядку n, вектори x і y з n елементами. Отримати вектор A (x + y).
21. Є квадратні матриці A, B, C порядку n. Отримати матрицю (A + B)xC.
22. Є квадратні матриці A і B порядку n. Отримати матрицю AB + C, де C матриця порядку n, елементи якої обчислюються за формулою: C[i,j]=i+j, i, j = 1, n.
23. Є матриця A розмірністю MxN. Отримати транспоновану матрицю.
24. Дана матриця A розмірністю MxN. Обчислити m-у степінь цієї матриці. (A^1=A, A^2=A^1*A, A^3=A^2*A і т.д.).
25. Є квадратна матриця A порядку n і вектор X (n). Непарні рядки матриці A замінити на X.
26. Є квадратна матриця A порядку n і вектор X (n). Парні стовпці матриці A замінити на X.
27. Є квадратна матриця A порядку n і вектор X (n). У матриці A поміняти місцями 1 і 2 рядки, 3 і 4, n і n-1 (скористатися X як допоміжним масивом).
28. Є квадратна матриця A порядку n. Визначити k кількість "особливих" елементів, вважаючи елемент "особливим", якщо він більше суми інших елементів свого стовпчика.
29. Є квадратна матриця A порядку n. Визначити k кількість "особливих" елементів, вважаючи елемент "особливим", якщо в його рядку ліворуч від нього знаходяться менші елементи, а праворуч - більші.

Приклад. Заповнити п’ятірками нижню частину квадратної матриці між діагоналями (діагоналі враховувати), проведеними через будь-який елемет a[k][m].

Лістінг 3.11

```
#include <stdio.h>
#include <stdlib.h>
#define N 7
#define M 7
void vyvid(int a[N][N], int n, int m);
/*  Робота з нижньою частиною квадратної матриці між
діагоналями (діагоналі враховувати), проведеними через будь-
який елемент */
int main()
{  int a[N][N]={0}; // заповнення матриці нулями
   int i, j; // індекси елементів масиву
   int jl, jp; // індекси колонок зліва і справа
   int n, // робочий розмір матриці
   k,m; // індекси вказаного елемента
   n=N-2;
   system("chcp 1251 & cls");
   printf("\nВкажіть порядок матриці n ");
   scanf("%d",&n);
   printf("\nЗадайте індекси елемента m і k ");
   scanf("%d %d",&m,&k);
   a[k][m]=5;
   jl=m; jp=m;
   /*  заповнюється п’ятірками частина матриці разом з
діагоналями */
   for (i=k; i<n; i++)
   { // якщо i=k+1, то без діагоналей
      for (j=jl; j<=jp; j++)
          a[i][j]=5;
      if (jl>0) jl--;
      if (jp<n-1) jp++; printf("\n");
   }
   printf("\nОброблена матриця:\n");
   for (i=0; i<n; i++)
   {  for (j=0; j<m; j++)
        printf("%5d",a[i][j]);
        printf("\n");
   }
   printf("\n\n");
   system("pause");
   return 0;
}
```

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Булгакова О. С. Алгоритмізація і програмування: теорія та практика : навчальний посібник для дистанційного навчання / О. С. Булгакова, В. В. Зосімов, Г. В. Ходякова. – Миколаїв: СПД Румянцева, 2021. – 138 с.
2. Трофименко О.Г. С++. Алгоритмізація та програмування : підручник / О.Г. Трофименко, Ю.В. Прокоп, Н.І. Логінова, О.В. Задерейко. 2-ге вид. перероб. і доповн. Одеса : Фенікс, 2019. 477 с. Основи інформатики та обчислювальної техніки: підручник / В. Г. Іванов, В. В. Карасюк, М. В. Гвозденко; за заг. ред. В. Г. Іванова. — Х.: Право, 2015. — 312 с.
3. Sarah L. Harris, David Harris. Digital Design and Computer Architecture: ARM Edition 1st Edition. – Morgan Kaufmann. – 2015. – 584p.
4. Іванов В. Г. Основи інформатики та обчислювальної техніки: підруч. / В. Г. Іванов, В. В. Карасюк, М. В. Гвозденко; зазаг. ред. В. Г. Іванова. – Х.: Право, 2012.
5. Sommerville I. Software Engineering, 10th ed. — Addison-Wesley / Pearson Education Limited, 2015. — 816 p.
6. Електроніка та мікросхемотехніка: підручник / О.М. Воробйова, І.П. Панфілов, М.П. Савицька, Ю.В. Флейта. – Одеса: ОНАЗ ім. О.С. Попова, 2015. – 298 с.
7. Albert Paul Malvino. Digital computer electronics. – New Delhi : Tata Mcgraw Hill Education Pvt. Ltd. – 2011. – 522 p.
8. James Lance. The Beginner's Guide to Engineering: Computer Engineering. - CreateSpace Independent Publishing Platform. – 2013. – 158p. ISBN-10 : 1492981540.
9. B. Stroustrup: A Tour of C++ (Second Edition). Addison-Wesley, 2018. - 240 pages.
10. G.L. McDowell: Cracking the Coding Interview. 6th Edition. 189 Programming Questions and Solutions. CareerCup, LLC, Palo Alto, CA. 2016. - 696 p