МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»
Кафедра кібербезпеки та математичного моделювання

ІНФОРМАТИКА

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
для здобувачів
першого (бакалаврського) рівня вищої освіти
освітньо-професійної програми «Кібербезпека»
спеціальності 125 Кібербезпека та захист інформації

Обговорено і рекомендовано
на засіданні кафедри
Кібербезпеки та математичного
моделювання
Протокол №2
від 13 лютого 2024 р.

Чернігів 2024

Інформатика. Методичні вказівки до виконання лабораторних робіт для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Кібербезпека» спеціальності 125 Кібербезпека та захист інформації. – Чернігів: НУ «Чернігівська політехніка», 2024 – 42 с.

Укладачі: ГРЕБЕННИК АЛЛА ГРИГОРІВНА, старший викладач кафедри кібербезпеки та математичного моделювання;
ПЕТРЕНКО ТАРАС АНАТОЛІЙОВИЧ, доцент кафедри кібербезпеки та математичного моделювання, кандидат технічних наук;
ДЮБА ІГОР МИКОЛАЙОВИЧ, викладач кафедри кібербезпеки та математичного моделювання

Відповідальний за випуск – ТКАЧ ЮЛІЯ МИКОЛАЇВНА,
завідувач кафедри кібербезпеки та математичного моделювання, доктор педагогічних наук, професор

Рецензент – МЕХЕД ДМИТРО БОРИСОВИЧ,
доцент кафедри кібербезпеки та математичного моделювання, кандидат педагогічних наук, доцент

ЗМІСТ

Вступ

Метою викладання навчальної дисципліни «Інформатика» є оволодіння студентами навичок алгоритмічного мислення, вивчення базових алгоритмів обробки простих структур даних, вироблення вмінь та навичок для самостійної розробки програмного забезпечення типових задач інформатики, отримання практичних навичок роботи за комп'ютером по налагодженню та тестуванню програм.

Виконання лабораторних робіт має на меті отримання практичних навичок розробки алгоритмів рішення різноманітних задач та їх реалізації мовами програмування високого рівня на прикладі мови С++. Для виконання робіт пропонується середовище розробки Visual Studio, що дозволяє створювати різні типи проектів, і в подальшому буде корисним при вивченні спеціальних предметів.

Цикл лабораторних робіт виконується протягом семестру і охоплює основні теми курсу «Інформатика». Теоретичною основою для виконання лабораторних робіт є курс лекцій, теоретичні відомості на початку кожної лабораторної роботи даних методичних вказівок та навчальна література.

Методичні вказівки до кожної лабораторної роботи виконані по єдиній структурі що включає мету роботи, теоретичну частину з прикладами реалізації програм за темою роботи, завдання для самостійного виконання та перелік контрольних питань.

Під час підготовки до лабораторного заняття студент має ознайомитися з темо та метою роботи, його робочим завданням, вивчити теоретичний матеріал, та, за можливості, розробити алгоритм рішення індивідуальних завдань, скласти текст програм, підготувати початкові (вхідні) дані для налагодження програм та виконання контрольних розрахунків.

Під час лабораторного заняття студент показує викладачеві результати роботи, консультується з питань, що виникли, та завершує роботу. По кожній роботі студент повинен оформити звіт в електронному вигляді за допомогою текстового редактора Word. Вимоги до вмісту звіту наведені в кожній лабораторній роботі.

Для захисту лабораторної роботи студент представляє звіт з виконання роботи . При оцінюванні роботи враховуються теоретичні знання за переліком контрольних питань та практичні вміння розв'язання простих задач за темою роботи або внесенні деяких змін у розроблені програми в присутності викладача. Бали, набрані студентом, дораховуються до модульних оцінок поточного контролю.

Успішне виконання та захист всіх лабораторних робіт є умовою допуску студента до складання іспиту.

Лабораторна робота №1. Функції. Передача параметрів.

Мета роботи: Розібратись з базовими поняттями програмування та отримати навички створення програм з використанням функцій.

Інструменти:

Для виконання лабораторної роботи вам знадобиться комп'ютер зі встановленим середовищем розробки Visual Studio, що дозволяє створювати різні типи проектів, але поки що ми будемо створювати тільки проекти C++ у вигляді консольних застосувань.

Консоль – це сукупність вікна на екрані та клавіатури. Інформація із програми виводиться у вікно, а введення інформації у програму здійснюється за допомогою клавіатури одночасно відображаючись на екрані.

Теоретична частина
Структура програми на мові C++

Програма на мові C++ – це файл, який має розширення .cpp (.c, якщо програма написана на C). Файл програми являє собою сукупність різноманітних оголошень (звернень до бібліотечних функцій, оголошення прототипів функцій, користувацьких типів, шаблонів структур, визначення макросів тощо) та функцій, одна з яких обов'язково з назвою main(). Це головна функція програми з якої починається її виконання.

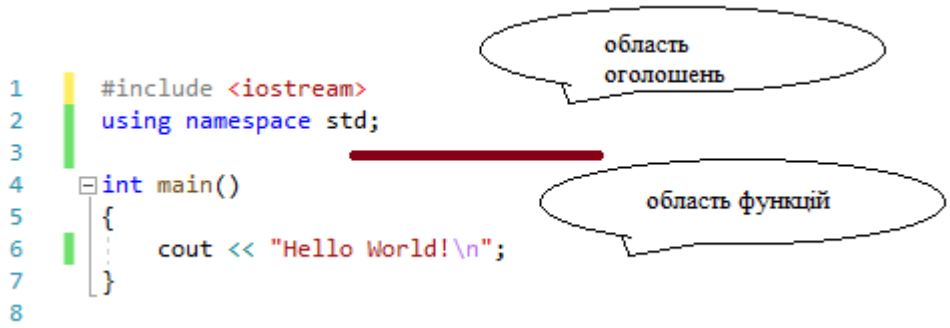


Рисунок 0.1 - Структура програми

Функція - це оформлений стандартним чином, логічно завершений фрагмент коду, що має власне ім'я, вирішує деяку локальну задачу і може повертати якийсь результат.

Рисунок 0.2 - Структура функції main()

Перший рядок функції є її заголовком, що складається з повідомлення про тип значення, яке повертає функція, назви функції та списку формальних параметрів у круглих дужках.

У більшості середовищ розробки очікується, що функція main() має повертати код завершення - зазвичай ціле число 0 повідомляє, що програма завершилася успішно. Це число повертає оператор return.

Тіло функції, що знаходиться у фігурних дужках, складається з описів та операторів, що реалізують дії, які повинна виконувати функція.

Функції

Функції є основними будівельними блоками програми. Вони необхідні коли в різних місцях програми необхідно виконати одну й ту ж саму послідовність дій (обчислити певну послідовність виразів, вивести одну й ту ж саму інформацію, тощо). Ці дії можуть бути досить складними і задаватися великими фрагментами програми. Для забезпечення компактності і підвищення наочності програм, усі логічно завершені її блоки рекомендується оформлювати у вигляді функцій (підпрограм).

Підпрограма - іменований, спеціальним чином оформлений, логічно завершений програмний фрагмент, який в процесі роботи програми може виконуватись багаторазово при звертанні до нього з різних місць програми.

Усі функції, що використовуються в програмі, повинні бути попередньо **оголошені**. Опис (як і для функції main()) складається із:

1) заголовка, що містить:
   - тип значення, що повертається,
   - ім’я функції,
   - список оголошень формальних параметрів, записаний у круглих дужках,
2) програмного блока (тіла функції), що описує дії, які вона виконує.

```
тип_значення ім’я_функції(список_формальних_параметрів)
{
  тіло_функції
}
```

Список формальних параметрів у заголовку функції являє собою оголошення відповідних змінних:

```
тип ім'я;
```

Формальні параметри являють собою змінні, через які в підпрограму передаються вхідні дані і, за потреби, повертаються результати. Ці дані передаються у неї як аргументи (фактичні параметри) під час виклику функції. Безпосередньо в термінах цих змінних і описуються дії в тілі функції, що робить їх загальними для різних вхідних даних. Якщо функція не містить параметрів, то список формальних параметрів – порожній список – () – тобто обов'язково залишаються круглі дужки.

Тіло функції виконуються тільки тоді, коли здійснюється звертання до даної функції яке називають **викликом** функції:

```
ім'я_функції(список_фактичних_параметрів)
```

Список фактичних параметрів являє собою послідовність виразів (може бути звичайне значення, змінна, вираз або виклик іншої функції), які задають значення формальних параметрів, що перелічені в описі функції. Фактичні параметри обов'язково мають відповідати формальним за порядком розташування, типом та кількістю (якщо не передбачена передача значень параметрів за замовчуванням).

Усі оголошення в тілі функції є локальними - доступними тільки всередині функції і тільки якщо звернення до них відбувається після оголошення. Отже, оператори тіла функції можуть використовувати формальні параметри, локальні змінні так і глобальні дані (що описані в зоні оголошень поза функціями).

При активізації (виклику) функції керування передається першому її оператору. Коли виконання завершується, відбувається повернення керування в основну програму, яка продовжує своє виконання з наступного за оператором виклику оператора.

Повернення значення із функції виконується оператором return:

```
return [вираз];
```

Результат обчислення виразу (його тип має співпадати з типом функції в її оголошенні) повертається функцією в місце її виклику. Якщо функція нічого не повертає (її тип void), оператор return записується без виразу, або взагалі не використовується.

Оператор return завершує роботу функції, незалежно від того де він розташований у тілі функції. За логікою алгоритму функції може використовуватися декілька операторів return.

Приклад функції яка не повертає результату:

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

void print_hello()      //оголошення функції
{
    cout << "Hello, world!\n";
}

int main()
{
    print_hello();      //виклик функції
}
```

Приклад функції яка повертає результат обчислення діагоналі прямокутника:

```cpp
#include <iostream>
#include <math.h>
using namespace std;

float diagonal(float a, float b)        //оголошення функції зі списком формальних параметрів
{
    return sqrt(pow(a, 2) + pow(b, 2)); // повернення результату обчислення виразу
}

int main()
{
    cout<< diagonal(1, 2);              //виклик функції з передачею фактичних параметрів
}
```

Декларація функції користувача у програмі C/C++ здійснюється або у вигляді оголошення функції, яке розташовують вище оголошення головної функції main(), або у вигляді *прототипу* функції.

Прототип функції – це попередній опис функції за допомогою тільки її заголовка, що надає компілятору можливість перевірити правильність звертання до функції та використання її результату. Використання прототипів підвищує читабельність коду: в області оголошень наводять прототипи функцій, потім функцію main(), а далі оголошення функцій, які послідовно розкривають алгоритм вирішення задачі.

Для визначення прототипу використовують заголовок функції:

```
тип_значення ім'я_функції(список_формальних_параметрів);
```

Наприклад для функції з попереднього прикладу прототип:

```
float diagonal(float a, float b);
```

Компілятор ігнорує ідентифікатори параметрів у прототипі функції, тому їх можна не вказувати. В такому разі попередній прототип набуває вигляду:

```
float diagonal(float, float);
```

Способи передачі параметрів у функції

Існує два способи передачі параметрів у функції - передача за значенням (by value) і передача через посиланням (by reference). Спосіб передачі вказується при оголошенні параметра у списку формальних параметрів.

За замовчуванням передбачається, що параметри звичайних типів, наприклад, float, double, int, char передаються за значенням, а параметри таких типів як масиви передаються через посилання. Якщо виникає необхідність вказати, що параметр передається через посилання, то перед ім'ям параметра, пишеться символ **&**.

Передача параметрів за *значенням* передбачає, що під час виклику функції у пам'яті буде виділена спеціальна область для запису копій значень фактичних параметрів, з якими і буде працювати функція, Це захищає змінні, передані у функцію в якості параметрів, від непередбачуваних змін, але вимагає додаткового обсягу пам'яті для створення копій змінних. У прикладі з діагоналлю прямокутника використано передачу параметрів за значенням.

Передача параметрів за *посиланням* передбачає, що до функції передаються адреси фактичних параметрів. Тому такий спосіб передачі називається ще передачею параметрів за адресом. Це заощаджує пам'ять і скорочує час звернення до функції, але будь-яка зміна формального параметру є зміною фактичного параметру, що робить дані більш вразливими.

Ще одна з переваг передачі параметрів через посилання – з функції можна повернути декілька результатів роботи функції. Наприклад використання передачі параметрів через посилання для повернення двох параметрів, що обмінюються значеннями.

```
void swap(float& a, float& b)
{
	float  temp = a;  a = b; b = temp;
}
```

Макроси з параметрами

Макроси з параметрами дають змогу здійснювати макропідстановки, подібні до викликів функцій. Формальні параметри, вказані в списку #define, під час препроцесування замінюються фактичними виразами, заданими у звертанні до макросу. Використання макросів замість функцій має дві переваги. По-перше, макрос вбудовується в тіло програми на етапі препроцесування, отже під час виконання програми не витрачається час на виклик функції. По-друге, аргументи, що вказуються у звертанні до макросів, можуть мати довільний тип.

Синтаксис макросу з параметрами такий:

```
#define ім'я_макросу(список_ параметрів) вираз_для_заміни
```

Декларація макросу для нашого прикладу з діагоналлю:

```
#define DIAGONAL(a, b) (sqrt(pow(a, 2) + pow(b, 2)))
```

Приклад звернення:

```
cout<< DIAGONAL(1, 2);
```

Приклад програми

Для прикладу розглянемо програму, що обчислює площу прямокутника за його сторонами. З головної функції програми main() викликаються три функції:

Init() – для ініціалізації вихідних даних;
Solution() – для розрахунку;
Browse() – для виведення результатів.

Уважно проаналізуйте порядок застосування прототипів, викликів та оголошень функцій та їх параметрів.

```
#include <iostream>
#include <Windows.h>
#include <math.h>
using namespace std;

//================= прототипи функцій =====================
void Init(float& a, float& b);
float Solution(float a, float b);
void Browse(float P);

//================= головна функція ========================
int main()
{
    SetConsoleOutputCP(1251);
    cout << "Обчислення площі прямокутника.\n";
    float a, b, P;
    Init(a, b);          //введення даних
    P=Solution(a,b);     //розрахунок площі прямокутника
    Browse(P);           //виведення результатів
}

//================ визначення функцій ======================
void Init(float & a, float & b)
{
    cout << "Введіть сторони прямокутника:\n";
    cout << "сторона a = ";
    cin >> a;
    cout << "сторона b = ";
    cin >> b;
}

float Solution(float a, float b)
{
    return a * b;
}

void Browse(float P)
{
    cout << "Площа прямокутника P = " << P;
}
```

Рисунок 0.3 – Приклад програми для обчислення площі прямокутника

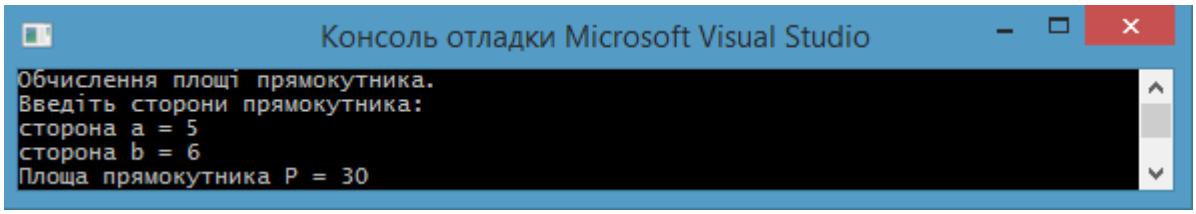


Рисунок 0.4 – Результат роботи програми з прикладу

ЗАВДАННЯ ДЛЯ ВИКОНАННЯ

Будемо визначати площу довільного трикутника S за його сторонами a, b, c за формулою Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

де $p = \frac{a+b+c}{2}$ - половина периметру трикутника або півпериметр.

Необхідно:

1. Скласти програму яка буде обчислювати площу трикутника. Введення даних, обчислення і виведення результату реалізувати у функції main().

2. Тепер виконай теж завдання з використанням окремих функцій для введення даних, обчислення і виведення результату що викликаються з функції main().

3. Створи макрос з параметрами для цього ж завдання та виведи на екран результат його виклику.

4. Проаналізуй як працюють функції другого завдання та макрос, якщо змінити тип аргументів (замість float передати значення типу int або double). Чи можна (потрібно) застосувати тут механізм явного приведення типів? Висновки зафіксуйте у звіті.

Вимоги до звіту

1. Назва роботи.
2. Мета роботи.
3. Завдання на лабораторну роботу.
4. Контрольний розрахунок (для контрольного набору вхідних даних виконати розрахунки проміжних і кінцевого результатів за допомогою калькулятора, щоб можна було перевірити результат роботи програми).
5. Тексти усіх файлів програм з коментарями.
6. Копії екранних форм результатів роботи.
7. Висновки.

Контрольні питання

1. Структура програми на C/C++.
2. Що таке функція та які переваги дає їх використання?
3. Чи допускається в програмах на C/C++ вкладеність функцій?
4. Прототип функції.
5. Оголошення функції.
6. Виклик функції.
7. Формальні параметри функцій.
8. Фактичні параметри функцій.
9. У чому полягає відмінність формальних параметрів від аргументів функції?
10. Як задати макрос з параметрами?
11. Відмінності в застосуванні макросів та функцій.
12. Локальні та глобальні змінні.

Лабораторна робота №2. Алгоритми з розгалуженнями

Мета роботи: вивчити особливості організації алгоритмів з розгалуженнями та набути навичок їх застосування при створенні програм.

Теоретична частина
Логічний тип даних

У мові С++ це логічний тип даних bool. Значення такого типу займає всього 1 байт пам'яті та використовується, насамперед, у логічних операціях, метою яких є визначення істинності або хибності виразу. Логічний вираз (або змінна логічного типу) може набувати тільки двох значень:

– false – хибність, 0;
– true – істина, 1 або будь-яке відмінне від нуля значення.

У мові С++ є 6 операторів порівняння:

Оператор	Символ	Приклад	Операція
Більше	>	x > y	true, якщо x більше y, в протилежному випадку – false
Менше	<	x < y	true, якщо x менше y, в протилежному випадку – false
Більше/Дорівнює	>=	x >= y	true, якщо x більше/дорівнює y, в протилежному випадку – false
Менше/Дорівнює	<=	x <= y	true, якщо x менше/дорівнює y, в протилежному випадку – false
Дорівнює	==	x == y	true, якщо x дорівнює y, в протилежному випадку – false
Не дорівнює	!=	x != y	true, якщо x не дорівнює y, в протилежному випадку – false

Оператори порівняння використовуються для перевірки тільки однієї конкретної умови: помилкова вона чи істинна. Коли потрібно протестувати відразу декілька умов необхідно застосувати логічні оператори.

У мові С++ є 3 логічних оператори:

Оператор	Символ	Приклад	Операція
Логічне НЕ	!	!x	true, якщо x – false і false, якщо x – true
Логічне І	&&	x && y	true, якщо x і y – true, в протилежному випадку – false
Логічне АБО	||	x || y	true, якщо x чи y – true, в протилежному випадку – false

Оператори розгалуження

Розгалуження в програмах можна реалізувати за допомогою:

– тернарного оператора ?;

– оператора розгалуження if;
– оператора вибору switch.

Тернарний оператор.

Його також називають умовною операцією або операцією вибору операнду. Назва походить від слова три оскільки операція виконується над трьома операндами. Синтаксис:

```
<умова> ? <вираз_1> : <вираз_2>;
```

Умова - це вираз, результат якого приводиться до логічного типу. Якщо результат обчислення умови приймає значення true, то результатом операції буде значення виразу_1. Якщо результат обчислення умови приймає значення false, то результатом операції буде значення виразу_2.

Наприклад:

```
(x<0)? "Число від'ємне" : "Число додатне";
```

Оператор розгалуження if.

Оператор if ... else дозволяє вибрати один з двох можливих варіантів виконання програми. Синтаксис:

```
if <умова>
    <оператор_1>;
else
    <оператор_2>;
```

Спочатку обчислюється значення умови (умова - вираз, результат якого приводиться до логічного типу). Якщо результат обчислення приймає значення true, то виконується оператор_1; інакше, якщо результат приймає значення false, то виконується оператор_2, що розташований за словом else. Якщо замість одного оператора потрібно виконати декілька, їх слід об'єднати у блок за допомогою фігурних дужок.

Наприклад:

```
if (x<0)
      cout<<"Число від'ємне";
else  cout<<"Число додатне";
```

У випадках коли при невиконанні умови не потрібно виконувати якоїсь дії, то оператор if можна застосовувати в скороченій формі:

```
if <умова>
   <оператор_1>;
```

Оператор if, як повний так і скорочений, може бути вкладений у інший оператор if. При цьому кожна else-частина пов'язується з найближчим if. Якщо ж else-частина відноситься до одного з попередніх if, то слід застосовувати фігурні дужки.

```
if (x<0)
      cout<<"Число від'ємне";
```

```
else if (x==0)
            cout<<"Число нуль";
      else cout<<"Число додатне";
```

Оператор вибору switch.

Оператор switch застосовують для реалізації численних розгалужень у випадках, коли вибір визначається значеннями змінної порядкового типу (int, char). Синтаксис:

```
switch (вираз)
    {
    case константа_1: оператори;
    case константа_2: оператори;
    ...
    case константа_N: оператори;
    default:          оператори;
    };
```

Спочатку обчислюється значення виразу. Зазвичай цим виразом є окрема змінна, але може бути і довільним виразом з цілочисловим значенням. Отримане значення послідовно порівнюється зі значеннями констант. Якщо значення виразу збігається з однією із міток, то виконується оператор цього варіанту і усі наступні за ним до default, якщо раніше не зустрівся оператор переривання break. Якщо значення виразу не збігається з жодною із міток, то виконується оператор після слова defaul, якщо ця частина присутня.

```
switch (month)
        {
        case 12:
        case 1:
        case 2:
            cout<<"Зима\n";
            break;
        case 3:
        case 4:
        case 5:
            cout<<"Весна\n";
            break;
        case 6:
        case 7:
        case 8:
            cout<<"Літо\n";
            break;
        case 9:
        case 10:
        case 11:
            cout<<"Осінь\n";
            break;
        default:
            cout<<"Такого місяця немає\n";
        };
```

Приклад програми

Для прикладу розглянемо програму, що визначає *існування* (якщо введені значення сторін більше 0) та *тип* прямокутника (квадрат чи звичайний прямокутник). З головної функції програми main() викликаються три функції:
Init() – для ініціалізації вихідних даних;
isRectangle() – для визначення існування фігури (повертає true – якщо існує, false – якщо не існує);
isType – для визначення типу (повертає 1 – якщо квадрат, 2 – якщо прямокутник);
Browse() – для виведення результатів.

```
#include <iostream>
#include <Windows.h>
#include <math.h>
using namespace std;

//================= прототипи функцій =====================
void Init(float&, float&);      //введення даних
bool isRectangle(float, float); //визначає чи існує прямокутник
int isType(float, float);       //визначає квадрат чи прямокутник
void Browse(int);               //виведення результатів

//================= головна функція ========================
int main()
{
    SetConsoleOutputCP(1251);

    float a, b;
    int type = 0;           //змінна типу int бо буде використовуватись у switch
    Init(a, b);             //введення даних
    if (isRectangle(a, b))  //якщо існує
        type=isType(a, b);  //визначаємо тип, присвоюємо його код змінній type
    Browse(type);           //виведення результатів
}

//=============== визначення функцій =======================

void Init(float& a, float& b)       //введення даних
{
    cout << "Введіть сторони прямокутника:\n";
    cout << "сторона a = ";
    cin >> a;
    cout << "сторона b = ";
    cin >> b;
}

bool isRectangle(float a, float b)  //визначає чи існує прямокутник
{
    if (a > 0 && b > 0)
        return true;    //існує
    else
        return false;   //не існує
}

int isType(float a, float b)        //визначає квадрат чи прямокутник
{
    if (a == b)
        return 1;   //1 - квадрат
    else
        return 2;   //2 - прямокутник
}

void Browse(int type)               //виведення результатів
{
    cout << "Тип - ";
    switch (type)
    {
    case 1:
        cout << "квадрат";
        break;
    case 2:
        cout << "прямокутник";
        break;
    default:
        cout << "не існує";
        break;
    }
}
```

Рисунок 2.1 – Приклад програми для визначення існування та типу прямокутника

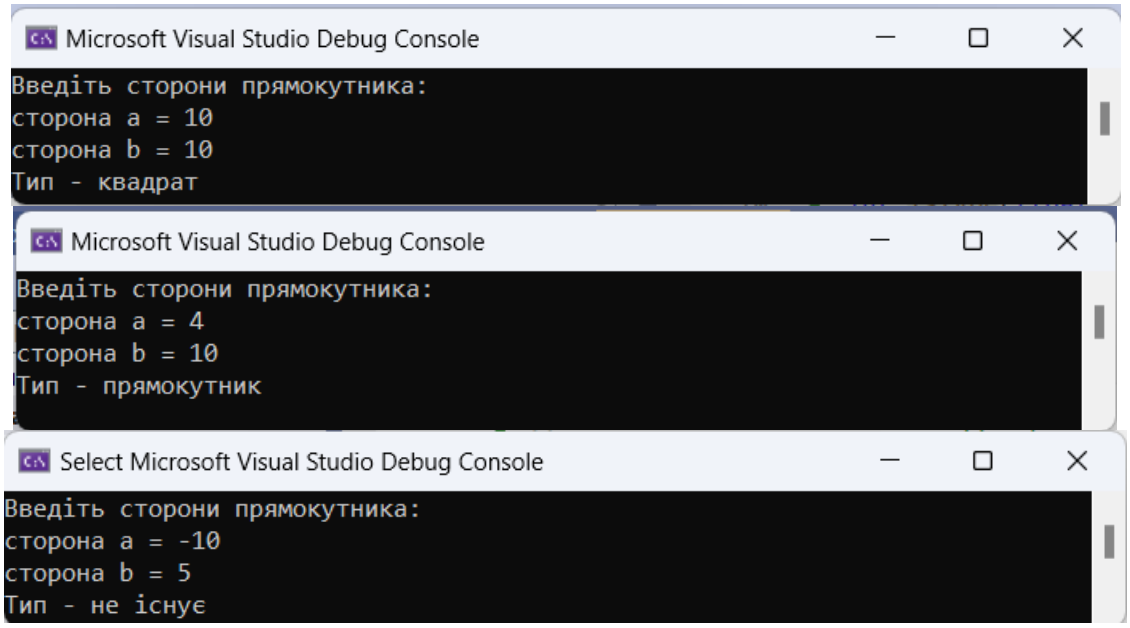


Рисунок 2.2 – Результати роботи програми з прикладу

ЗАВДАННЯ ДЛЯ ВИКОНАННЯ

Будемо визначати *існування* та *тип* трикутника:

Трикутник існує тоді і тільки тоді, коли сума довжин будь-яких двох його сторін більша за третю. При цьому довжина кожної сторони більше 0.

Тип трикутника *за кількістю рівних сторін*:

1. *Різносторонній* трикутник – всі три сторони не рівні.
2. *Рівнобедрений* трикутник – дві сторони рівні.
3. *Рівносторонній* трикутник або правильний трикутник – всі три сторони рівні.

Тип трикутника *за величиною кутів* (використовуємо наслідки з теореми косинусів https://www.youtube.com/watch?v=WC3Unu6rJl4):

4. *Гострокутний* трикутник - всі кути трикутника гострі (Якщо квадрат найбільшої сторони трикутника менший за суму квадратів двох інших сторін, то трикутник гострокутний).
5. *Тупокутний* трикутник - один з кутів трикутника тупий (Якщо квадрат найбільшої сторони більший за суму квадратів двох інших сторін, то трикутник тупокутний).
6. *Прямокутний* трикутник - один із кутів трикутника прямий (Якщо квадрат найбільшої сторони дорівнює сумі квадратів двох інших сторін, то трикутник прямокутний).

Необхідно:

1. Скласти програму яка буде визначати існування та тип трикутника. Введення даних, обчислення і виведення результату реалізувати у функції main().

2. Тепер виконай теж завдання з використанням окремих функцій що викликаються з функції main():

- для введення даних;
- визначення існування трикутника;
- визначення типу трикутника за кількістю рівних сторін;
- визначення типу трикутника за величиною кутів;
- виведення результату за допомогою оператора switch.

Вимоги до звіту

1. Назва роботи.
2. Мета роботи.
3. Завдання на лабораторну роботу.
4. Контрольний розрахунок (для контрольного набору вхідних даних, щоб можна було перевірити результат роботи програми).
5. Тексти усіх файлів програм з коментарями.
6. Копії екранних форм результатів роботи.
7. Висновки.

Контрольні питання

1. Логічний тип даних.
2. Оператори порівняння.
3. Логічні оператори та обчислення їх результатів.
4. Вирази з логічними операторами та правила їх обчислення.
5. Оператор **if** та його варіанти.
6. Тернарний оператор.
7. Оператор **switch.**

Лабораторна робота №3. Циклічні алгоритми

Мета роботи: вивчити особливості організації циклічних алгоритмів набути навичок їх застосування при створенні програм.

Теоретична частина
Оператори циклу

Оператори циклу використовують для здійснення багаторазового повторення деякої послідовності дій. Кожен цикл містить *тіло циклу*, тобто оператори, що виконуються декілька разів. Один прохід циклу називається *ітерацією*. У мові C++ існують три оператори циклу: **while**, **do while**, **for**. Оператори while зручно застосовувати у випадках, коли кількість ітерацій заздалегідь не відома, а for – коли кількість повторень відома.

Оператор циклу з передумовою while передбачає перевірку умови повторення циклу перед кожною його ітерацією і має вигляд:

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

де оператор — тіло циклу, що може бути представлено простим або складеним оператором (блок операторів розміщений між фігурними дужками {}).

Реалізується оператор while таким чином: якщо значення виразу-умови не дорівнює нулю (true), то виконується тіло циклу, а в протилежному випадку, тобто коли значення виразу дорівнює нулю (false), - цикл не працює і керування передається наступному за циклом while оператору. Цикл з передумовою може не виконуватися жодного разу.

Як вираз-умова використовуються логічні вирази або змінна логічного типу. Слід зауважити, що для того щоб цикл завершився, потрібно щоб послідовність операторів тіла циклу впливала на значення виразу-умови.

Наприклад необхідно обрахувати суму чисел від 1 до 10 за допомогою оператора while:

```
i=0, sum=0;
while (i<10)
{
    i++;
    sum +=i;
};
cout<<"Сума "<<sum;
```

Оператор циклу з післяумовою do while зазвичай застосовується у випадках коли тіло циклу необхідно виконати хоча б один раз і має таку форму запису:

```
do
    оператор
while (вираз-умова);
```

У процесі виконання оператора do while спочатку здійснюється вхід до тіла циклу і виконується оператор, що являє собою тіло циклу (цей оператор може бути простим або складеним); далі перевіряється вираз-умова і, якщо він правдивий (true), — цикл повторюється, а коли вираз помилковий (false) — здійснюється вихід з циклу.

Наприклад таж сама сума чисел від 1 до 10 з використанням оператору do while:

```
int i=0, sum=0;
do {
    i++;
    sum +=i;
} while (i<10);
cout<<"Сума "<<sum;
```

Оператор циклу з параметром for має форму запису вигляд:

```
for (вираз1; вираз2; вираз3)
оператор;
```

де вираз1 — вираз ініціювання, що використовується для встановлення початкового значення параметра, це вираз присвоювання;
вираз2 — вираз-умови, що визначає умову повторення циклу;
вираз3 — вираз ітерації, який визначає крок зміни параметра, що керує циклом, після кожного виконання.

Вирази вираз1, вираз2 та вираз3 — необов'язкові параметри, які розділяються символом «;».

Оператор циклу for реалізується таким чином:

- виконується вираз ініціювання (виконання цієї нотації може бути здійснено до оператора for);
- обчислюється вираз-умова;
- якщо умовний вираз приймає значення «true» — виконуються оператори циклу;
- обчислюється вираз ітерації;
- знову перевіряється умова;
- як тільки умова прийме значення «false», керування передається оператору, що розташований за оператором циклу for.

Оператор for може використовувати декілька змінних, що керують циклом, а будь-які вирази можуть бути відсутніми. Для суми чисе від 1 до 10 цикл for можна записати як:

```
for(int i=1; i<=10; i++)
    sum+=i;
```

або:

```
for(sum=0, i=1; i<=10; i++, sum+=i);
```

Вкладеність циклів

Всі оператори циклів допускають вкладеність інших операторів, у тому числі і інших циклів. Якщо до складу циклу входить інший цикл, то має місце вкладеність циклів. При цьому перший цикл називається *зовнішнім*, а вкладений у нього – *внутрішнім*. Внутрішній цикл виконується повністю під час кожної ітерації зовнішнього циклу.

Кожний з пари вкладених циклів має свою керуючу змінну і свої параметри. При виконанні вкладених циклів діє правило: у першу чергу завжди виконується самий внутрішній цикл. Таким чином, для кожного значення керуючої змінної зовнішнього циклу керуюча змінна внутрішнього циклу послідовно пробігає усі свої значення.

Усередині вкладеного циклу може знаходитися ще один вкладений цикл. Тобто той самий цикл може бути зовнішнім стосовно одного і внутрішнім стосовно іншого циклу. Границі внутрішнього циклу не можуть виходити за границі зовнішнього циклу.

В програмі можна використовувати будь-які комбінації вкладених циклів всіх типів, якщо цього потребує логіка побудови програми.

Наприклад для формування таблиці множення можна використати два вкладені цикли for:

```
for (int i=1; i<=10; i++) {
    for (int j=1; j<=10; j++)
        cout << i*j;
    cout << "\n";
}
```

Оператори переривання циклів

Щоб достроково перервати виконання *циклу* (до аналізу умови), можна використати оператор переривання **break**. Даний оператор може використовуватися в циклах всіх трьох типів.

Виконання оператора break призводить до виходу з циклу, в якому він знаходиться, і переходу до наступного за циклом оператора. Якщо оператор break знаходиться всередині вкладених циклів, то його дія поширюється тільки на той цикл, в якому він безпосередньо знаходиться.

Для дострокового переривання *ітерації циклу* служить оператор продовження **continue**. Цей оператор призводить до переходу до наступної ітерації *без завершення поточної*. Він, як і оператор break, може використовуватися тільки серед операторів тіла циклу.

Оператор continue, як і оператор break, перериває цикл в якому він знаходиться.

Крім цих операторів можна використовувати оператор **return**, який перериває виконання не тільки циклу, але і всієї функції у разі якщо за логікою програми цикл оформлено окремою функцією

Безкінечні цикли

Безумовні або безкінечні цикли, це цикли які повторюються безкінечну кількість разів. Часто використовуються на практиці, коли потрібно виконати невідому на початку циклу кількість операцій. Для виходу з циклів використовують оператор **break**.

Варіанти реалізації:

```
while(true)
{
    //тіло циклу
}

for(;;)
{
    //тіло циклу
}

do
{
    //тіло циклу
}
while(true);
```

Приклад програми

Приклад 1. Обчислення з точністю до «машинного нуля» значення суми числового ряду

$$\frac{1}{1\cdot 2\cdot 3}+\frac{1}{2\cdot 3\cdot 4}+\frac{1}{3\cdot 4\cdot 5}+\frac{1}{4\cdot 5\cdot 6}+\cdots$$

```
#include <iostream>
#include <windows.h>
#include <cmath>
#include <iomanip> // для setprecision()

using namespace std;

int main(int argc, char *argv[])
{
    SetConsoleOutputCP(1251);
    // обчислення суми ряду з точністю до "машинного нуля"
    double sum1, sum2; // попередня й поточна суми
    int n; // номер ітерації
    n=0;
    sum2=0.;
    do {
        n++;
        sum1=sum2;
        sum2 +=1.0/(n*(n+1)*(n+2));
    } while (sum1<sum2);
    cout<<"Сума "<<n<<" членів ряду \n\t1/(1*2*3)+1/(2*3*4)+... = "<<setprecision(16)<<sum2;

    return a.exec();
}
```

```
Сума 1290 членів ряду
        1/(1*2*3)+1/(2*3*4)+... = 0.24999969930345
```

Рисунок 3.1 – Приклад програми для визначення суми числового ряду

Приклад 2. Обчислити квадратний корінь з будь-якого дійсного числа a.

Треба розглянути такі випадки:
— якщо $a < 0$, то квадратний корінь не існує;
— якщо $a = 0$, то корінь дорівнює 0;
— якщо $a > 0$, то для обчислення квадратного кореня застосувати метод Ньютона (метод дотичних для знаходження нулів функції з заданим початковим наближенням):

$$b_0 = 1; \qquad b_{n+1} = \frac{\frac{a}{b_n} + b_n}{2},$$

де b_n - наближене значення $\sqrt{a}$.

Для виходу із циклу ітерацій (послідовних наближень) треба скористатися відносною похибкою:

$$\left| \frac{a}{b_n^2} - 1 \right| < 10^{-6}$$

```
#include <iostream>
#include <windows.h>
#include <cmath>
#include <iomanip> // для setprecision()

using namespace std;

int main(int argc, char *argv[])
{
    SetConsoleOutputCP(1251);

    /* Обчислення квадратного кореня за методом дотичних Ньютона */
    const double eps=1.e-6;
    double num, bnum; // введене число та наближене значення кореня
    cout<<"Введіть дійсне число: ";
    cin>>num;
    if (num<0) cout<<"\tКорінь не існує";
    else
        if (num==0) cout<<"\t0";
    else { // число >0
        bnum=1;
        do
                bnum=(num/bnum+bnum)/2;
        while (fabs(num/(bnum*bnum)-1)>=eps);
        cout<<"\nКорінь квадратний = " << bnum;
    }

    return a.exec();
}
```

```
Введіть дійсне число: 258

Корінь квадратний = 16.0624
```

Рисунок 3.2 – Приклад програми для визначення кореня квадратного за формулою Ньютона

ЗАВДАННЯ ДЛЯ ВИКОНАННЯ

Запрограмувати ітераційні процеси, використовуючи оператори циклів.

1. Знайти найбільший спільний дільник (НСД) двох чисел, використовуючи алгоритм Евкліда: від більшого числа віднімаємо менше, доки числа не стануть рівними — це і є НСД.

2. Обчислити значення числа π за двома формулами:

$$\pi = 3 + 4\left(\frac{1}{2\cdot 3\cdot 4} - \frac{1}{4\cdot 5\cdot 6} + \frac{1}{6\cdot 7\cdot 8} - \cdots\right) = 3 + \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n(2n+1)(n+1)};$$

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots\right) = 4\cdot \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{2n-1}.$$

Порівняти швидкість збіжності — кількість ітерацій (доданків) для досягнення заданої точності ε порявняно з бібліотечним числом π (константа M_PI бібліотеки math.h).

3. Обчислити корінь ступеня k з будь-якого дійсного числа a за методом дотичних Ньютона. Для врахування можливих варіантів значення ступеня k і значення числа a запрограмувати такі вкладені умови:

- якщо $k = 0$, то корінь знайти не можливо;
- якщо $a < 0$ і k — парне або $a = 0$ і k — від'ємне, то корінь не існує;
- якщо $k = 1$ або $a = 0$, то результатом є a;
- якщо $k = -1$, то результатом є

$$\frac{1}{a};$$

- якщо $k < 0$, то шукати

$$\frac{1}{\sqrt[k]{a}}.$$

Для обчислення кореня ступеня k застосувати метод Ньютона:

$$b_0 = 1;$$

$$b_{n+1} = \frac{\dfrac{a}{b_n^{k-1}} + (k-1)b_n}{k}$$

де b_n - наближене значення $\sqrt[k]{a}$.

Для виходу із циклу ітерацій (послідовних наближень) треба скористатися відносною похибкою:

$$\left|\frac{a}{b_n^k} - 1\right| < 10^{-6}$$

Вимоги до звіту

1. Назва роботи.
2. Мета роботи.

3. Завдання на лабораторну роботу.
4. Тексти усіх файлів програм з коментарями.
5. Копії екранних форм результатів роботи.
6. Висновки.

Контрольні питання

1. Циклічні оператори.
2. Оператора **while**. Правила виконання.
3. Оператора **do … while**.. Правила виконання.
4. Оператора **for**. Правила виконання.
5. Безкінечні цикли.
6. Оператори переривання циклів **break** та **continue**.
7. Вкладеність циклів.

Лабораторна робота №4. Одновимірні масиви. Алгоритми сортування та пошуку.

Мета роботи: вивчити особливості організації одновимірних масивів, набути навичок програмної реалізації алгоритмів сортування та пошуку.

Теоретична частина
Поняття масиву

Масив - це структура даних, що являє собою впорядкований набір фіксованої кількості *однотипних* компонент (елементів масиву) довільної структури. Масив може бути одновимірним, або багатовимірним, тобто таким у якого кожний елемент є також масивом.

В оперативній пам’яті комп’ютера масив займає зв’язну область пам'яті в якій зберігаються елементи масиву без проміжків між ними. Ім’я масиву зберігає адресу першого елементу цього масиву, тобто є вказівником на його початок.

Масив визначається *ім'ям*, *кількістю елементів* та *їх типом*. Одновимірний масив в С++ оголошується наступним чином:

```
тип_елементів ім’я [кількість_елементів];
```

Наприклад,

```
int a[3];
double b[10];
```

Кожен з елементів масиву має свій порядковий номер, що зветься індексом. Нумерація елементів починається з 0. Характерною особливістю масивів є прямий доступ до їх елементів: всі компоненти масиву можуть вибиратися довільно і однаково доступні. Для позначення окремого елемента в цьому випадку потрібно, окрім імені, вказати його порядковий номер (індекс):

```
ім'я [індекс]
```

В якості індексів можуть використовуватися довільні вирази, що включають константи і змінні допустимих порядкових типів. Наприклад:

```
m[5];
k[i+5];
```

Приклад функції виведення масиву на екран:

```
void print_array(int array[], int size) {
    for(int i = 0; i<size; i++) {
        cout<<array[i]<<' ';
    }
}
```

У С++ доступ до елементів масиву може здійснюватися також за покажчиком, оскільки ім’я масиву – покажчик на його перший елемент. Таким чином, якщо маємо оголошення масиву `int a[10]` то звертання `*a` дає той самий результат, що й `a[0]`, а звертання `*(a+9)` дасть той самий результат, що й `a[9]`.

Над покажчиками можливі такі операції:

– присвоєння - за допомогою операції присвоєння покажчику можна присвоїти значення адресної константи, адресу якоїсь змінної або результат обчислення виразу, що знаходиться праворуч від знака присвоєння (необхідною умовою операції присвоєння для покажчиків є однаковість базових типів вказівника і значення, що йому присвоюється);
– порівняння - можна використовувати звичайні операції порівняння (найчастіше використовуються операції == та !=);
– збільшення/зменшення - до значення покажчиків можна додавати (або віднімати) цілі числа (при цьому значення покажчика змінюється на величину числа помножену на кількість байтів, що займає у пам'яті елемент даних відповідного типу). При обробці масивів найчастіше використовують операції інкременту та декременту які зміщують покажчик до наступного або попереднього елемента масиву відповідно;
– віднімання - повертає кількість елементів базового, типу що може розміститися між адресами, на які вказують перший та другий операнди операції.

Приклад функції, що заповнює масив випадковими числами, а звернення до елементів масиву реалізується через вказівники:

```
void create_array_random(int array[], int size) {
    srand(time(NULL) | clock());        //запуск генератора випадкових чисел
                                        //необхідні бібліотеки <time.h> та <cstdlib>
    int* end = array + size;
    for (int* p = array; p < end; p++)
    {
        *p = rand() % 100;              //генерується число в діапозоні від 0 до 99
    }
}
```

Обробка масивів

Обробка елементів одновимірних масивів, зазвичай, виконується в циклах `for`, які забезпечують послідовне звертання до кожного елемента масиву. При цьому, як правило, індекси елементів масиву використовуються як параметри циклу.

Базовими операціями обробки одновимірних масивів є:
– введення масиву, його ініціалізація;
– виведення масиву;
– пошук максимального або мінімального елемента масиву;
– обчислення узагальнюючих характеристик (сум елементів, їх добутків, тощо);
– пошук заданого елемента;
– перестановка елементів або обмін значеннями між елементами масиву;
– вставка та видалення елемента масиву.

Базові операції обробки масивів слід реалізовувати у вигляді функцій, які згодом можуть бути використані як «архітектурні блоки» при розв'язанні більш складних задач.

У С++ передача масиву у функцію здійснюється тільки за посиланням (адресом). Особливості передачі масиву параметром функції:

1) Оскільки масиви можуть мати різну розмірність, необхідно передавати у функцію два параметри - покажчик на масив (ім'я масиву) і кількість його елементів, наприклад:

```
int mas1[10], mas2[100];
float f(int arr[], int size)
{ …
}
int main()
{ cout<<f(mas1,n)<<" "<<f(mas2, m);
}
```

2) У разі необхідності заборони модифікації значень масиву використовується специфікація `const`, наприклад:

```
float f(const int arr[])
```

3) Окремі елементи масиву передаються у функцію за значенням, наприклад:

```
float f_max(int a, int b)
{ …
}
int main()
{ …
cout<< f_max(arr[i],arr[j]);
}
```

Алгоритми сортування масивів

Сортування – один з найбільш розповсюджених процесів обробки даних, що здійснює упорядковане за певним правилом розміщення об'єктів. Існують різні методи сортування, що відрізняються швидкістю отримання результату, складністю і універсальністю.

Сортування масиву – це процес перестановки елементів масиву з метою їх розміщення в певному порядку. Наприклад, якщо сортується масив чисел за зростанням їх значень, то кожен наступний елемент має бути більше попереднього; якщо сортується масив чисел за спаданням їх значень, то кожен наступний елемент має бути менше попереднього; якщо сортується масив символів або рядків, то елементи розміщуються в алфавітному порядку тощо.

Далі розглянемо три найпростіші методи сортування масивів:

1. Метод вибору

Алгоритм сортування:

1. Серед усіх елементів масиву, починаючи з першого, шукають найменший (найбільший) елемент.
2. Знайдений найменший елемент міняють місцями з першим елементом.
3. Переглядають масив від другого елементу, і знаходять найменший серед цих елементів.

4. Знайдений найменший елемент міняють місцями з другим елементом.
5. Проходимо масив стільки разів, скільки *елементів-1*.

Для програмної реалізації цього методу сортування буде потрібно два цикли. У *зовнішньому* циклі повинен змінюватися номер елемента, куди буде заноситися черговий найменший елемент. Номери цих елементів мають змінюватися від першого до передостаннього. Цей цикл буде визначати кількість проходів по масиву. *Внутрішній* цикл повинен забезпечити послідовне порівняння елемента, зафіксованого першим циклом, з усіма елементами, які слідують в масиві за ним. Якщо в результаті порівняння знаходиться елемент, що менший ніж зафіксований, то порівнювані елементи міняються місцями.

Наприклад, сортування числового масиву за *зростанням* їх значень:

```
void Sort_0(int array[], int size) {
    int tmp;
    for (int i=0; i<size-1; i++)
        for (int j=i+1; j<size; j++)
            if (array[j] < array[i])
            {
                tmp=array[i];
                array[i]=array[j];
                array[j]=tmp;
            }
}
```

Для сортування того ж масиву за *спаданням* значень достатньо змінити умову в операторі if з array[j] < array[i] на array[j] > array[i].

Наведена функція не єдиний варіант реалізації методу вибору. Для зменшення кількості операцій перезапису можна у внутрішньому циклі тільки запам’ятовувати індекс найменшого елемента, а сам обмін робити після завершення внутрішнього циклу.

2. Метод обміну (метод бульбашки)

Алгоритм заснований на принципі порівняння сусідніх елементів та обміні значеннями, якщо їх розташування не відповідає правилу впорядкування масиву. Процес починається з першого елемента і закінчується передостаннім. Після першого проходу, найбільший елемент виявиться на останньому місці, бо де б він не був спочатку, після кожного порівняння він буде зміщуватися праворуч. При цьому менші елементи змістяться на одну позицію ліворуч.

У наступному проході зовнішнього циклу останній елемент перевіряти вже не потрібно, тому процес завершується перед впорядкованою частиною масиву, а наступний найбільший серед переглянутих елементів виявляється на передостанньому місці.

Далі процес повторюється. Оскільки кожен прохід по масиву від його початку упорядковує щонайменше один елемент у хвості масиву, тому потрібна кількість проходів по масиву дорівнює його розміру -1.

```
void Sort_1(int array[], int size) {
    int tmp;
    for (int k=size-1; k>0; k--)
        for (int i=0; i<k; i++)
            if (array[i+1] < array[i])
            {
                tmp=array[i];
                array[i]=array[i+1];
                array[i+1]=tmp;
            }
}
```

Розглянута функція робить проходи по масиву незалежно від результатів сортування. Навіть, якщо масив вже впорядкований, функція буде робити задану кількість проходів. Ознакою впорядкованості масиву на деякій ділянці є відсутність обмінів. Тому для скорочення кількості проходів по масиву, можна зафіксувати індекс останнього елемента, який приймав участь в обміні. В наступному проході має сенс розглядати тільки ті елементи, які мають менший індекс.

3. Метод вставки

На кожному кроці елементи масиву поділені на впорядковану частину, яка розташовується на початку масиву, та невпорядковану решту масиву, і перший елемент з невпорядкованої частини вставляється в упорядковану. Пошук місця для вставки відбувається шляхом послідовного зсуву праворуч елементів впорядкованої частини.

На початку сортування впорядкована частина складається всього з одного, першого елемента, а всі інші елементи розташовуються в другій частині масиву.

Послідовні кроки алгоритму сортування полягають у тому, що перший елемент з невпорядкованої частини порівнюється з останнім елементом впорядкованої послідовності. Якщо виявляється, що порядок розташування цих елементів не відповідає вимогам сортування, то елемент з невпорядкованої частини запам’ятовується і на його місце зсувається останній елемент впорядкованої частини.

Зсув упорядкованих елементів на одну позицію вправо триває, поки не буде знайдено місце для елемента, вилученого з невпорядкованої послідовності.

Функція сортування має два цикли. У зовнішньому циклі послідовно змінюється номер лівої межі невпорядкованою області від другого елементу (i=1) до кінця масиву. У тілі цього циклу порівнюються елементи, що знаходяться по обидва боки від межі, що розділяє впорядковану і невпорядковану частини. Якщо порядок порушений, то перший елемент невпорядкованою послідовності запам’ятовується у змінній tmp, внаслідок чого звільняється місце для зсувів упорядкованих елементів вправо. Внутрішній цикл do.while забезпечує послідовні зсуви упорядкованих елементів вправо, починаючи з останнього, поки не буде знайдено місце для елемента, що зберігався у змінній tmp.

```
void Sort_2(int array[], int size) {
    int tmp, j;
    for (int i=1; i<size; i++)
        if (array[i] < array[i-1])
        {
        tmp=array[i];
        j=i;
        do {
            array[j]=array[j-1];
            j--;
        } while (j>0 && array[j-1]>tmp);
        array[j]=tmp;
    }
};
```

Ефективність роботи цієї функції можна дещо підвищити за рахунок зменшення кількості порівнянь під час пошуку місця для елементу із буфера. Місце вставки для елементу із буфера у впорядкованій частині можна знайти методом дихотомії.

Злиття впорядкованих масивів

З двох впорядкованих масивів можна сформувати третій, також впорядкований, масив без додаткового сортування. Ця операція є основою алгоритму сортування злиттям, але поки розглянемо сам алгоритм злиття двох впорядкованих масивів.

Цей алгоритм починається з порівняння *перших* елементів вихідних масивів array1 і array2. За лічильником i будемо вибирати елементи з масиву array1, за лічильником j — з масиву array2, а за лічильником k — заносити елементи до масиву array3. У масив array3 заноситься менший з елементів, що порівнюються, а далі в порівнянні бере участь наступний елемент того масиву, елемент якого вже записаний до array3. Ця процедура повторюється, доки елементи одного з масивів не закінчяться. Елементи, що залишилися в іншому масиві, окремим циклом перезаписуються в результуючий масив.

```
void Join(int array1[], int size1, int array2[], int size2, int array3[], int size3) {
    int i, j, k;
    for (i = 0, j = 0, k = 0; i < size1 && j < size2; k++)
    {
        if (array1[i] < array2[j])
            array3[k] = array1[i++];
        else
            array3[k] = array2[j++];
    }
    if (i == size1)
        for (; j < size2;)
            array3[k++] = array2[j++];
    else
        for (; i < size1; )
            array3[k++] = array1[i++];
}
```

Алгоритми пошуку елементів в масиві

Алгоритми пошуку – це клас задач мета яких визначити наявність та індекс шуканого елемента в масиві.

Для невпорядкованих масивів застосовується алгоритм *лінійного* (*послідовного*) пошуку. Він перебирає всі елементи масиву, перевіряючи їх на співпадіння з шуканим елементом:

```
int linSearch(int arr[], int element, int size)
{
    for (int i = 0; i < size; i++)
    {
        if (arr[i] == element)  //елемент знайдено
            return i;
    }
    return -1;                  //елемент не знайдено
}
```

Для невпорядкованого масиву даних, пошук неможливо прискорити – час перебору завжди буде в лінійній залежності від розміру масиву. Особливо якщо потрібно знайти всі (не тільки перший) індекси шуканого значення – потрібно завжди перебрати усі елементи один за одним. Ситуація змінюється, коли ми знаємо, що масив даних певним чином впорядкований.

Пошук у впорядкованому масиві можна прискорити за рахунок перебору частини елементів масиву де шуканий елемент повинен знаходитись за правилом сортування. Тут також існує множина алгоритмів пошуку. Далі розглянемо один з найуживаніших – алгоритм бінарного пошуку (дихотомії).

Бінарний пошук - це ефективний алгоритм пошуку елемента у відсортованому масиві. Його реалізація полягає в тому, що на кожному кроці алгоритму ми порівнюємо шуканий елемент з елементом, що знаходиться в середині масиву. Якщо шуканий елемент менший за середній елемент, то ми продовжуємо пошук в лівій половині масиву, інакше - в правій. Таким чином, на кожному кроці ми зменшуємо кількість елементів, серед яких потрібно шукати, вдвічі. Цикл пошуку переривається коли елемент знайдено, або закінчується коли діапазон області пошуку звузиться до нуля, що буде свідчити про те, що потрібного елементу в масиві немає. Приклад функції бінарного пошуку:

```
int bi_Search(int arr[], int element, int size)
{
    int left = 0, right = size - 1, m;  //початкова область пошуку - всі елементи масиву
    do
    {
        m = left + (right - left) / 2;  // визначаємо середину області пошуку
        if (arr[m] == element)          // елемент знайдено в позиції m
            return m;
        if (arr[m] < element)
            left = m + 1;               // посуваємо ліву межу вправо від елементу масиву
        else
            right = m - 1;              // посуваємо праву межу вліво від елементу масиву
    } while (left <= right);
    return -1;                          //елемент не знайдено
}
```

Приклад програми

Створимо 3 масиви. Перший і другий заповнимо випадковими числами та відсортуємо методом вибору. Третій – функцією злиття двох сортованих масивів заповнюється елементами першого та другого масиву. Потім введімо значення елемента для пошуку та виконаємо функцію пошуку цього елементу для кожного з масивів.

Нижче наведено фрагмент програми з прототипами усіх використаних функцій та головною функцією main(). Реалізації всіх необхідних функцій наведені вище.

```cpp
#include <iostream>
#include <Windows.h>
#include <math.h>
#include <cstdlib>
#include <time.h>
using namespace std;

//================== прототипи функцій ======================
void create_array_random(int array[], int size);    //заповнення масиву випадковими числами
void print_array(int array[], int size);            //виведення масиву на екран
void Sort_0(int array[], int size);                 //сортування методом вибору
int bi_Search(int arr[], int element, int size);     //бінарний пошук в сортованому масиві
void Join(int array1[], int size1, int array2[], int size2, int array3[], int size3);   //злиття відсортованих масивів

//================== головна функція ========================
int main()
{
    SetConsoleOutputCP(1251);
    int size1 = 10, size2 = 10, size3 = 20;
    int ar1[10], ar2[10], ar3[20];

    cout << "\nЗгенерований перший масив: \n";
    create_array_random(ar1, size1);
    print_array(ar1, size1);
    cout << "\nВідсортований перший масив: \n";
    Sort_0(ar1, size1);
    print_array(ar1, size1);

    cout << "\nЗгенерований другий масив: \n";
    create_array_random(ar2, size2);
    print_array(ar2, size2);
    cout << "\nВідсортований другий масив: \n";
    Sort_0(ar2, size2);
    print_array(ar2, size2);

    cout << "\nРезультат злиття двох відсортованих масивів: \n";
    Join(ar1, size1, ar2, size2, ar3, size3);
    print_array(ar3, size3);

    cout << "\nВведіть значення для пошуку: \n";
    int element, pos;
    cin >> element;

    pos = bi_Search(ar1, element, size1);
    if (pos == -1)
        cout << "В першому масиві елемент " << element << " не знайдено.\n";
    else
        cout << "В першому масиві елемент " << element << " знайдено в позиції " << pos << ".\n";

    pos = bi_Search(ar2, element, size2);
    if (pos == -1)
        cout << "У другому масиві елемент " << element << " не знайдено.\n";
    else
        cout << "У другому масиві елемент " << element << " знайдено в позиції " << pos << ".\n";
    pos = bi_Search(ar3, element, size3);
    if (pos == -1)
        cout << "У другому масиві елемент " << element << " не знайдено.\n";
    else
        cout << "У другому масиві елемент " << element << " знайдено в позиції " << pos << ".\n";
}

```

Рисунок 4 .1 – Фрагмент прикладу програми з сортуванням та пошуком елементів в масивах

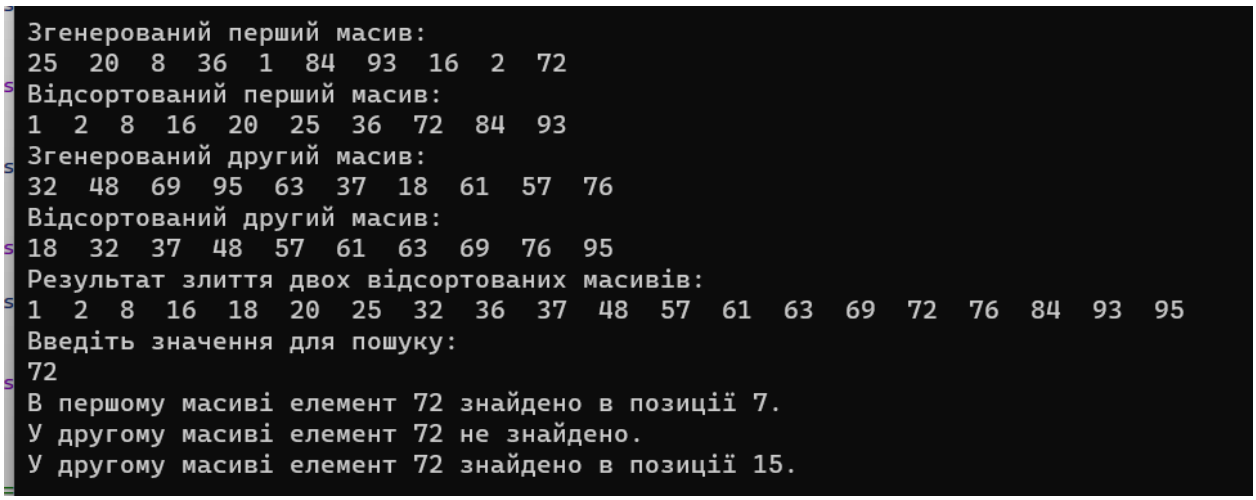


Рисунок 4 .2 – Результат роботи прикладу програми з сортуванням та пошуком елементів в масивах

ЗАВДАННЯ ДЛЯ ВИКОНАННЯ

Створіть 3 масиви та реалізуйте наступні процеси їх обробки:

1. Реалізуйте функцію виведення елементів масиву на екран, та використайте її для візуалізації результатів роботи наступних процесів.
2. Перший масив заповніть випадковими числами.
3. Перший масив відсортуйте методом обміну (бульбашки).
4. Другий масив заповніть самостійно введенням чисел з клавіатури.
5. Другий масив відсортуйте методом вставки.
6. Третій масив заповніть елементами першого та другого масиву функцією злиття двох сортованих масивів заповнюється.
7. Реалізуйте алгоритм тернарного пошуку. Його реалізація подібна до реалізації бінарного пошуку, але тут на кожному кроці алгоритму ми порівнюємо шуканий елемент з двома елементами, що знаходяться в середині масиву на рівновіддалених позиціях $m1$ та $m2$. Для першої ітерації пошуку визначаємо межі пошуку – весь масив: l – індекс першого елементу масиву, r – індекс останнього елемент масиву, $m1 = l + (r - l)/3, m2 = r - (r - l)/3$. Якщо:
   - шуканий елемент менший за елемент в позиції $m1$, то ми продовжуємо пошук в лівій третині масиву,
   - якщо шуканий елемент більший за елемент в позиції $m2$, то ми продовжуємо пошук в правій третині масиву,
   - інакше - в середній третині.

   Таким чином, на кожному кроці ми зменшуємо кількість елементів, серед яких потрібно шукати, втрічі. Цикл пошуку переривається коли елемент знайдено, або закінчується коли діапазон області пошуку звузиться до нуля, що буде свідчити про те, що потрібного елементу в масиві немає.
8. Введіть значення елемента для пошуку та визначте його наявність в усіх трьох масивах.

Вимоги до звіту

1. Назва роботи.
2. Мета роботи.
3. Завдання на лабораторну роботу.
4. Тексти усіх файлів програм з коментарями.
5. Копії екранних форм результатів роботи.
6. Висновки.

Контрольні питання

1. Визначення поняття масив.
2. Оголошення масиву і його ініціалізація.
3. Масиви як параметри функцій.
4. Особливості обробки масиву за допомогою вказівників.
5. Алгоритми сортування масивів.
6. Алгоритми пошуку елементів в масиві.
7. Вставка та видалення елементів з масиву.
8. Об’єднання впорядкованих масивів.

Лабораторна робота №5. Робота з рядками. Шифрування.

Мета роботи: вивчити особливості організації символьних масивів, набути навичок програмної реалізації алгоритмів обробки рядків тексту.

Теоретична частина
Оголошення рядків

Змінна (або константа), охоплена подвійними лапками, розглядається, як масив символів. Для такого масиву в оперативній пам'яті виділялася ділянка, розмір якої на один байт більший, ніж кількість символів у рядку. Цей додатковий байт використовується для збереження ознаки кінця рядка. У якості такої ознаки використовується символ '\0' (має код, що дорівнює 0). За розташуванням нуль-символа визначається фактична довжина рядка або кількість елементів символьного масиву. Вона дорівнює кількості символів у рядку плюс 1, тому що нуль-символ також є елементом масиву.

Для оголошення рядка використовуються звичайні засоби опису масивів. Індексування такого масиву, як і будь-якого іншого, починається з нуля, а ім'я масиву є покажчиком на його перший елемент. Наприклад:

```
char <ім'я рядка> [<кількість символів >];
char str[80];
char str[] = "Hello!";
```

Зверніть увагу, що ми можемо оголосити фактичний розмір масиву або залишити квадратні дужки порожніми, якщо одразу ініціалізуємо масив. Оскільки масив символьний, то значення самого рядка утримується в подвійні лапки.

Для роботи з рядками в мові C++ також є стандартна бібліотека шаблонів (STL), яка містить клас string з засобами обробки рядків та дозволяє оголошувати змінні типу string:

```
#include <string>
string str;
string str = "Hello!";
```

Під час виконання програми рядок символів можна ввести з консолі за допомогою об'єкту cin так само, як і число. Але cin зчитує рядок до перших пробілу або ознаки кінця рядка. Тому для зчитування з консолі рядків, що містять пробіли, краще використовувати метод getLine об'єкту cin:

```
cin.getline(str, 80);
```

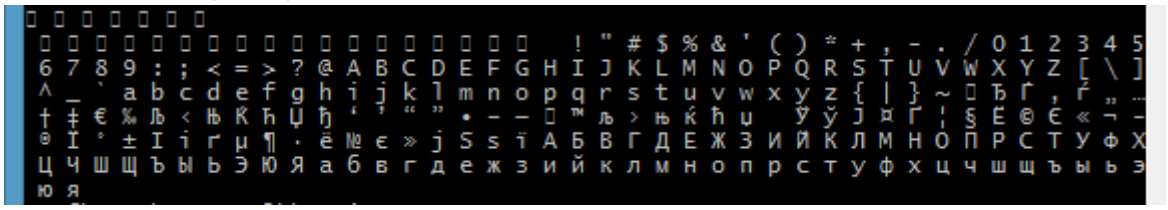
Перший параметр функції getline – змінна в якій буде збережене введене значення, другий - найбільша кількість символів, яку можна записати в рядок.

Коди символів

Всі символи, які доступні комп'ютеру, утворюють так звану таблицю символів. Кожний символ має в таблиці свій номер, який називається його ASCII-кодом (американський стандартний код для обміну інформацією).

Дізнатися символ за його кодом можна шляхом приведення значення коду до типу char. Наприклад виведемо на друк перші 256 символів таблиці кодування:

```
for (int i=0; i<256 ;i++)
    cout << (char) i << " ";
```



Великі англійські, малі англійські літери, літери кирилиці та цифри в кодовій таблиці упорядковані, тобто сусідні коди відрізняються на 1.

Дізнатися код символу можна шляхом приведення його значення до типу int.

Наприклад визначимо коди символів введеного рядка:

```
char firstStr[80];
cout << "Введіть рядок символів : ";
cin.getline(firstStr,100);
cout << "\nВи ввели рядок символів: " << firstStr;
cout << "\nКоди введених символів : ";
for (int i=0; firstStr[i]!='\0';i++)
{
    cout << "\nСимвол : " << firstStr[i] << "  має код : "<< (int) firstStr[i];
}
```

```
Введіть рядок символів : АБВГД abcde

Ви ввели рядок символів: АБВГД abcde
Коди введених символів :
Символ : А  має код : -64
Символ : Б  має код : -63
Символ : В  має код : -62
Символ : Г  має код : -61
Символ : Д  має код : -60
Символ :    має код : 32
Символ : a  має код : 97
Символ : b  має код : 98
Символ : c  має код : 99
Символ : d  має код : 100
Символ : e  має код : 101
```

```
Введіть рядок символів : Hello рядок

Ви ввели рядок символів: Hello рядок
Коди введених символів :
Символ : H  має код : 72
Символ : e  має код : 101
Символ : l  має код : 108
Символ : l  має код : 108
Символ : o  має код : 111
Символ :    має код : 32
Символ : р  має код : -16
Символ : я  має код : -1
Символ : д  має код : -28
Символ : о  має код : -18
Символ : к  має код : -22
```

В цьому прикладі цикл оброблює кожен елемент символьного масиву поки поточний символ не стане рівним '\0'. Також умовою продовження циклу може бути довжина рядка обчислена аналогічним циклом або спеціальними функціями обробки символьних масивів.

Обробка рядків символів

Рядки можуть оброблятися як цілісний об'єкт, так і посимвольно. При посимвольній обробці доступ до конкретного символу рядка здійснюється за індексом або за покажчиком масиву символів.

Для обробки рядка як цілісного об’єкта зазвичай необхідно використовувати цикли, що перебирають усі елементи масиву як було наведено у прикладі вище.

При роботі з рядками часто необхідно виконати операції з:

- перевірки символів;
- перетворення символів;
- перевірки рядків;
- маніпулювання рядками.

Мова C++ має декілька бібліотек для обробки символьних типів даних функцій з великою кількістю вбудованих функцій:

- string.h (https://uk.wikipedia.org/wiki/String.h)
- ctype.h (https://uk.wikipedia.org/wiki/Ctype.h)
- stdlib.h (https://ru.wikipedia.org/wiki/Stdlib.h)

Ознайомтеся з переліком цих функцій за посиланнями.

Шифрування

Обробка рядків лежить в основі процесів шифрування, що використовують криптографічні ключі та математичні алгоритми. Простий метод криптографічного перетворювання, заснований на правилі переставляння літер у відкритому тексті.

Для прикладу розглянемо один з найпростіших методів - шифр Цезаря з ключем 1. В основі методу лежить ідея заміни символів повідомлення на інші за допомогою ключа. Оскільки в прикладі ключ дорівнює 1, то кожний символ необхідно замінити на наступний за ним в алфавітному порядку. Таким чином символ ‘a’ перетворюється в символ ‘b’, символ ‘b’ в ‘c’, …, символ ‘z’ в ‘a’. Для заміни символу на наступний достатньо збільшити значення його коду на 1, а для переходу на початок алфавіту – застосувати умовні оператори. Символи, що не належать до алфавіту, в шифрованому тексті залишаються незмінними. Нижче наведено фрагмент програми, що шифрує символи латиниці та кирилиці рядку str шифром Цезаря з ключем 1:

```
char str[]="Be sure to drink your Ovaltine";
for (int i=0; str[i]!='\0';i++)
{
    switch (str[i])
    {
    case 'z':
        str[i]='a'; break;
    case 'Z':
        str[i]='A'; break;
    case 'я':
        str[i]='a'; break;
    case 'Я':
        str[i]='A'; break;
    default: if(str[i]>='a'&& str[i]<'z' ||
                str[i]>='A'&& str[i]<'Z' ||
                str[i]>='a'&& str[i]<'я' ||
                str[i]>='A'&& str[i]<'Я')
            str[i]++;
    }
}
cout<<str;      Cf tvsf up esjol zpvs Pwbmujof_
```

Шифри перестановки відносяться до симетричних. Це означає, що один і той же алгоритм, а також один і той же ключ використовуються і для шифрування, і для дешифрування повідомлень. Шифри перестановки мають невелику криптостійкість, тому їх не використовують без додаткових перетворювань або використовують методи асиметричного чи гібридного шифрування.

ЗАВДАННЯ ДЛЯ ВИКОНАННЯ

1. Завдання – “Initials”.

Напишіть програму, що запитує ім’я користувача, а як результат обробки виводить його ініціали у верхньому регістрі без пробілів або крапок. Вважайте, що ввід користувача містить лише літери (верхнього та/або нижнього регістрів) з пробілами. Наприклад:

введений рядок:	результат:
Zamyla Chan	ZC
robert thomas bowden	RTB
максим петренко	МП

2. Завдання “Аве, Цезарю!”.

Шифр Цезаря або шифр зсуву — симетричний моноалфавітний алгоритм шифрування, в якому кожна буква відкритого тексту заміняється на ту, що віддалена від неї в алфавіті на сталу кількість позицій (Шифр Цезаря — Вікіпедія (wikipedia.org).

Римський імператор Юлій Цезар використовував для приватного листування шифр зсуву з ключем 3 (замість літери A підставляв D, замість B - E і так далі).

Іншими словами, якщо:
p - деякий звичайний незашифрований текст,
p_i - i-й символ в p,
k - ключ (невід'ємне ціле число),
c – шифрований текст,
c_i – i-та літера в шифрованому тексті c, що обчислюється таким чином:

$$c_i = (p_i + k) \% n$$

де n — потужність (кількість букв алфавіту) алфавіту.

Неалфавітні символи, такі як знаки пунктуації, пробіли та цифри, не змінюються.

Наприклад, є секретний ключ $k = 13$ та звичайний текст p англійського алфавіту (n=26):

Be sure to drink your Ovaltine

Тоді зашифрований текст c отримуємо з p та k, зміщенням кожної літери тексту p на 13 позицій відповідного алфавіту:

Or fher gb qevax lbhe Binygvar

Зверніть увагу на те, що літера O (перша літера в зашифрованому тексті) зміщена на 13 позицій від літери B (перша літера в незашифрованому тексті). Так само й r (друга в зашифрованому тексті) на 13 більша за e (друга в незашифрованому тексті). В той же час, літера f (третя літера в зашифрованому тексті) на 13 позицій менша за s (третя літера в початковому тексті), в цьому випадку ми переходимо по кругу від z до a.

Шифр Цезаря з ключем 13, як правило, називають ROT13 (http://en.wikipedia.org/wiki/ROT13). Насправді, краще використовувати ROT26, який, як вважають, в два рази безпечніший.

Ваше завдання - написати програму, що шифрує повідомлення за допомогою шифру Цезаря з ключем 13. Текст може бути в англійському алфавіті кирилицею.

Вимоги до звіту

1. Назва роботи.
2. Мета роботи.
3. Завдання на лабораторну роботу.
4. Тексти усіх файлів програм з коментарями.
5. Копії екранних форм результатів роботи.
6. Висновки.

Контрольні питання

1. Оголошення рядка символів і його ініціалізація.
2. Як позначається кінець рядка у C++?
3. Як може здійснюватися доступ до елемента рядка?
4. Бібліотечні функції для змінних рядкового типу.

5. Рядки символів як параметри функцій.

Рекомендована література

1. Костюк І.В., Козак Л.І., Стасевич С.П. Основи програмування — Новий світ-2000, 2021р., - 328 с.
2. Васильєв О.М. Програмування на С++ в прикладах та задачах — Ліра К, 2019р., - 382 с.
3. Шпак З.Я. Програмування мовою С / З.Я. Шпак. – Львів: вид-во НУ «Львівська політехніка», 2011. – 436с.