МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чернігівський національний технологічний університет

ОСНОВИ ПРОГРАМУВАННЯ НА С, С++

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторного практикуму та самостійної роботи з дисципліни
«Програмування»
для студентів напряму підготовки
6.050103 – "Програмна інженерія"

ЗАТВЕРДЖЕНО
на засіданні кафедри
програмної інженерії
протокол № 1 від 31.08.15

Чернігів ЧНТУ 2015

Основи програмування на С, С++. Методичні вказівки до лабораторного практикуму та самостійної роботи з дисципліни «Програмування» для студентів напряму підготовки 6.050103 – “Програмна інженерія” /Укл.: Бивойно П.Г., Бивойно Т.П. – Чернігів: ЧНТУ, 2015. – 153 с.

Укладачі: Бивойно Павло Георгійович, кандидат технічних наук, доцент
Бивойно Тарас Павлович, старший викладач

Відповідальний за випуск: В.В. Литвинов, зав. кафедрою програмної інженерії, доктор технічних наук, професор

Рецензент: С.О. Нестеренко, кандидат технічних наук, доцент кафедри інформаційних і комп'ютерних систем Чернігівського національного технологічного університету

ЗМІСТ

1 Лабораторна робота № 1. Знайомство з інтегрованим середовищем розробкм (ICP)“QT-creator” 9
1.1 КОРОТКІ ВІДОМОСТІ ПРО ICP “QT-creator” 9
1.1.1 Головне меню середовища 10
1.1.2 Панель режимів робіт 10
1.1.3 Панель доступу до результатів опрацювання проекту 10
1.1.4 Створення проекту 10
1.1.5 Панель редагування програм 11
1.1.6 Функція main() 13
1.1.7 Ще один приклад простої програми 14
1.1.8 Збереження проекту 15
1.1.9 Етапи виконання програми 15
1.2 Вимоги до звіту 15
1.3 Контрольні питання 15
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 15
2 Лабораторна робота № 2. Типи даних та розрахунки за формулами 16
2.1 Короткі теоретичні відомості 16
2.1.1 Змінні і константи 16
2.1.2 Типи даних 16
2.1.3 Арифметичні типи даних 17
2.1.4 Описи змінних 20
2.1.5 Операція розміру sizeof 21
2.1.6 Арифметичні операції 21
2.1.7 Операції присвоєння 22
2.1.8 Вирази 23
2.1.9 Пріоритети операцій у C++ 24
2.1.10 Узгодження типів 24
2.1.11 Бібліотека математичних функцій cmath 25
2.1.12 Використання об’єкту cin для введення даних 26
2.1.13 Зміна кольорів консолі та керування курсором 27
2.2 Завдання на лабораторну роботу 29
2.3 Вимоги до звіту 29
2.4 Контрольні питання 30
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 30
3 Лабораторна робота № 3. Функції 31
3.1 Короткі теоретичні відомості 31
3.1.1 Правила написання функцій 31
3.1.2 Виклик функції 32
3.1.3 Прототип функції 33
3.1.4 Прототипи бібліотечних функцій 33
3.1.5 Способи передачі параметрів у функції 34
3.1.6 Області оголошення та доступу до імен 35
3.1.7 Макроси з параметрами 37

3.2 Реализація проекту «function» 37
3.2.1 Підключення допоміжних файлів 38
3.2.2 Створення макросу з параметрами 38
3.2.3 Створення прототипів функцій 39
3.2.4 Функція main() програмного файлу 39
3.2.5 Функція f1(), що повертає значення 40
3.2.6 Функція f2() типу void 40
3.2.7 Функція з виведенням проміжних результатів 41
3.3 Дослідження створеного проекту 41
3.3.1 Дослідження передачі параметрів за посиланням 41
3.3.2 Анализ передачи параметров по значению 42
3.4 Вимоги до звіту 42
3.5 Контрольні питання 42
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 42
4 Лабораторна робота № 4. Логічний тип даних і розгалуження у програмах 43
4.1 Короткі теоретичні відомостіI 43
4.1.1 Логічний тип даних 43
4.1.2 Алгоритми з розгалуженнями 44
4.1.3 Програмування розгалужень 46
4.1.4 Оператор переходу goto 50
4.2 Реализація проекту «if_switch» 51
4.2.1 Початковий інтерфейс проекту 51
4.2.2 Допоміжні файли 51
4.2.3 Прототипи функцій 52
4.2.4 Функція main() програмного файлу 52
4.2.5 Функція обробки номеру варіанта 53
4.2.6 Функція введення коефіцієнтів рівняння за вибором користувача . 53
4.2.7 Функція для розв’язання лінійного рівняння 53
4.2.8 Функція для розв’язання квадратного рівняння 54
4.3 Вимоги до звіту 54
4.4 Контрольні питання 54
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 54
5 Лабораторна робота № 5. Побудова циклів з операторми „while” і „do...while” 55
5.1 Короткі теоретичні відомості 55
5.1.1 Циклічні алгоритми 55
5.1.2 Оператор while 55
5.1.3 Оператор do…while 57
5.1.4 Переривання циклу 58
5.1.5 Ітераційні алгоритми 59
5.1.6 Алгоритми обчислення сум нескінченних рядів 60
5.2 Створення проекту «while_do» 63
5.2.1 Початковий інтерфейс проекту 63
5.2.2 Допоміжні файли 64

5.2.3 Прототипи функцій 64
5.2.4 Функція main() програмного файлу 65
5.2.5 Функція обробки номеру варіанта 65
5.2.6 Функції для обчислення кубічного кореня 66
5.2.7 Функції для обчислення синуса 67
5.3 Завдання для самостійної роботи 68
5.4 Вимоги до звіту 68
5.5 Контрольні питання 68
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 69
6 Лабораторная работа № 6. Обробка данных за допомогою циклу for 70
6.1 Короткі теоретичні відомостіI 70
6.1.1 Оператор циклу for 70
6.1.2 Особливості використання циклу for 71
6.1.3 Випадкові числа 72
6.1.4 Табулювання функцій 73
6.2 Завдання на лабораторну роботу 74
6.3 Приклад Створення проекту «цикл for» 75
6.3.1 Головна функція проекту 76
6.3.2 Табулювання функції 76
6.3.3 Обробка послідовностей цілих чисел 78
6.3.4 Обробка послідовності випадкових чисел 79
6.4 Вимоги до звіту 80
6.5 Контрольні питання 81
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 81
7 Лабораторна робота № 7. Одновимірні масиви 82
7.1 Короткі теоретичні відомості про масиви 82
7.1.1 Оголошення одновимірного масиву та звернення до його елементів 82
7.1.2 Приклад використання одновимірного масиву 84
7.1.3 Масиви символів 84
7.1.4 Одновимірні масиви як параметри функцій 85
7.2 Функції обробки масивів чисел 86
7.2.1 Функція формування випадкового масиву 86
7.2.2 Функції виведення масиву на консоль 87
7.2.3 Функції введення масиву з консолі 87
7.2.4 Функція введення масиву по елементам 89
7.2.5 Функція вилучення елементу з масиву 90
7.2.6 Функція перевороту масиву 90
7.2.7 Функція формування масиву накопичених значень 91
7.3 Функції обробки рядків символів 91
7.3.1 Функція копіювання частини рядка 91
7.3.2 Функція знаходження підрядка у рядку 92
7.4 Завдання для самостійної роботи 92
7.5 Вимоги до звіту 93
7.6 Контрольні питання 94

РЕКОМЕНДОВАНА ЛІТЕРАТУРА 94
8 Лабораторна робота № 8. Сортування масивів 95
8.1 Методи сортування масивів 95
8.1.1 Сортування вибором 95
8.1.2 Сортування обміном (метод бульбашки) 99
8.1.3 Сортування вставкою 104
8.2 Сортування за ускладненими правилами 109
8.3 Обробка упорядкованих масивів 110
8.3.1 Пошук позиції елемента у впорядкованому масиві 110
8.3.2 Вставка елементу до впорядкованого масиву 111
8.3.3 Видалення елементу з упорядкованого масиву 113
8.3.4 Злиття двох впорядкованих масивів 113
8.4 Завдання для самостійної роботи 114
8.5 Вимоги до звіту 115
8.6 Контрольні питання 115
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 116
9 Лабораторна робота № 9. Двовимірні масиви (матриці) 117
9.1 Короткі теоретичні відомості 117
9.1.1 Оголошення та ініціалізація матриць 117
9.1.2 Матриці як параметри функцій 118
9.1.3 Формування та виведення матриць з використанням консолі 119
9.1.4 Тотальна обробка даних у матрицях 121
9.1.5 Вибіркова обробка матриць 121
9.1.6 Перестановки елементів матриці 122
9.1.7 Видалення та вставка елементів матриці 123
9.1.8 Сортування елементів матриці 123
9.2 Завдання для самостійної роботи 127
9.3 Вимоги до звіту 129
9.4 Контрольні питання 129
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 130
10 Лабораторна робота № 10. Робота із структурами 131
10.1 Короткі теоретичні відомості 131
10.1.1 Оголошення шаблону та ініціалізація структур 131
10.1.2 Масиви структур 133
10.1.3 Введення-виведення структур 134
10.1.4 Сортування масивів структур 134
10.2 Створення проекту «Результати атестації» 134
10.2.1 Інтерфейс користувача для проекту 135
10.2.2 Визначення глобальних типів даних програми 135
10.2.3 Функція обробки номеру вибраного варіанту 136
10.2.4 Функція відображення масиву на консолі 136
10.2.5 Додавання нових даних до масиву структур 137
10.2.6 Функція сортування масиву за групою та прізвищем 137
10.2.7 Сортування за кількістю незадовільних оцінок та середньому балу 138

10.2.8 Вибірка студентів, що мають середній бал вище 4 139
10.2.9 Підрахунок кількості студентів що мають більше 2-х незадовільних оцінок 139
10.3 Завдання для самостійної роботи 139
10.4 Вимоги до звіту 141
10.5 Контрольні питання 141
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 141
11 Лабораторна робота № 11. Основи роботи з покажчиками 142
11.1 Короткі теоретичні відомості 142
11.1.1 Оголошення та ініціалізація покажчиків 142
11.1.2 Звернення до даних через покажчики 143
11.1.3 Використання кваліфікатора const для покажчиків 144
11.1.4 Адресна арифметика 144
11.1.5 Нетипізовані вказівники 145
11.2 Завдання для самостійної роботи 146
11.3 Вимоги до звіту 147
11.4 Контрольні питання 147
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 147
12 Лабораторна робота № 12. Використання покажчиків для обробки масивів та структур 149
12.1 Короткі теоретичні відомості 149
12.1.1 Приклад формування одновимірного масиву з використанням покажчиків 149
12.1.2 Робота з масивами символів через покажчики 150
12.1.3 Стандартні функції для роботи з рядками символів 151
12.1.4 Масиви покажчиків на рядки символів 151
12.1.5 Покажчики на структури 152
12.2 Завдання для САМОСТІЙНОЇ РОБОТИ 152
12.3 Вимоги до звіту 152
12.4 Контрольні питання 153
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 153

Вступ

Лабораторні роботи слугують сполучною ланкою між практичними заняттями та самостійною роботою студентів, під час якої виконується розрахунково-графічна робота. В процесі виконання лабораторних робіт експериментально перевіряються ключові питання курсу програмування, набуваються практичні навички побудови та налагодження програм, перевіряється ступінь засвоєння основних положень предмета. Під час лабораторних занять студенти знайомляться з типовими рішеннями деяких задач програмування.

Лабораторні роботи виконуються на персональних комп'ютерах в системі програмування Qt creator. Передбачається, що студенти знайомі з основами роботи на персональному комп'ютері. Необхідно володіти клавіатурою і мишкою, вміти маніпулювати з файлами, знати який-небудь редактор текстів. Якщо таких навичок немає, то студент повинен придбати їх під час самостійної роботи в лабораторії. Можна скористатися методичними вказівками «Основи роботи на персональному комп'ютері» для студентів економічних спеціальностей. Передбачається також, що студенти володіють англійською мовою в обсязі програми середньої школи.

Студент зобов'язаний до лабораторного заняття прочитати методичні вказівки до лабораторної роботи і спробувати виконати її самостійно. Під час лабораторного заняття студент показує викладачеві результати роботи, консультується з питань, що виникли, та завершує роботу. Обсяг виконаної роботи може бути різним, залежно від того, на яку оцінку претендує студент. Коли робота закінчена, студент повинен захистити її. Захист полягає у відповідях на питання по темі лабораторної роботи і внесенні деяких змін у розроблений проект, в присутності викладача.

По кожній роботі студент повинен оформити звіт. Звіти оформляються за допомогою текстового редактора Word на папері формату А4, у відповідності з вимогами стандартів на оформлення технічної документації. Звіт по роботі є розділом підсумкового документа. В кінці семестру звіти зшиваються в єдиний підсумковий документ з титульним листом, підписуються у керівника, після чого студент отримує допуск до іспиту.

За лабораторну роботу студент може отримати до 100 балів, з урахуванням своєчасності та якості виконання всіх складових роботи. Складовими є: звіт, проект і відповіді на контрольні питання. Оцінки, отримані за лабораторні роботи, враховуються при виставленні підсумкової оцінки. Для отримання допуску до іспиту всі роботи повинні бути виконані і кожна з них оцінена не менше ніж в 60 балів.

1 ЛАБОРАТОРНА РОБОТА № 1.
ЗНАЙОМСТВО З ІНТЕГРОВАНИМ СЕРЕДОВИЩЕМ РОЗРОБКМ (ICP)“QT-CREATOR”

Мета роботи:

– Отримати навички роботи з основними вікнами ICP.
– Створити найпростіший проект в ICP.

1.1 КОРОТКІ ВІДОМОСТІ ПРО ICP “QT-CREATOR”

Бібліотека Qt являє собою набір класів C++ та інструментів розробки програм для різних платформ і є безумовним лідером серед наявних засобів розробки мовою C++ програм, придатних для різних платформ.

Існує декілька варіантів середовища для роботи з цією бібліотекою. Повна версія, яку можна знайти в мережі Інтернет, досить важка і потребує потужного комп’ютера. Існують також спрощені версії, які менш вибагливі до ресурсів комп’ютера. Незважаючи на те, що кожна з таких версій має свої особливості, основні елементи середовища розробки у цих версіях однакові. Одну із таких версій Вам буде запропоновано у лабораторії.

ICP "Qt creator" запускається шляхом вибору іконки "Qt creator" на робочому столі, або через меню "ПУСК" і підменю " Програми ".

Після запуску на екрані з'являються вікно, що зображено на рисунку 1.1.

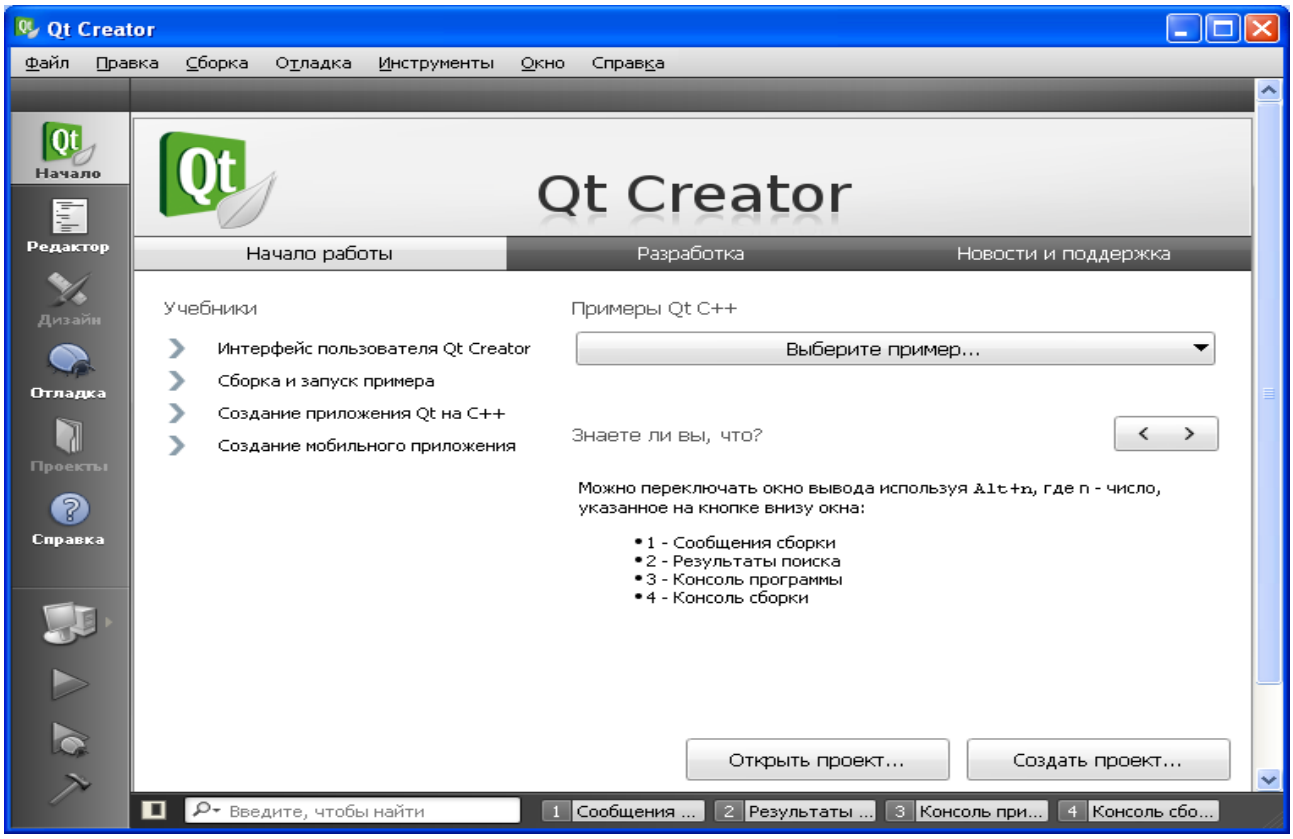


Рисунок 1.1 – Стартове вікно ICP "Qt creator"

1.1.1 Головне меню середовища

У верхній частині вікна, розташовано головне меню. Воно забезпечує роботу з файлами, редагування, запуск проекту на компіляцію та виконання, операції з вікном. У меню «Інструменти» є функція «Параметри», яка викликає діалог налаштування параметрів середовища. У цьому діалозі можна вибрати мову середовища, та таблицю кодування файлів. Для української мови найбільше підходить таблиця “ibm866/CP866/csIBM866”. На жаль ця таблиця не підтримує українську букву «і», тому її доведеться міняти на англійську.

1.1.2 Панель режимів робіт

Нижче меню, ліворуч розташована панель режимів робіт із середовищем. Кожному з режимів відповідає своя панель, що з’являється після вибору режиму.

На рисунку 1.1 у центральній частині вікна розташована панель режиму «Початок». На цій панелі є три закладки: «Початок роботи», «Розробка» та «Новини і підтримка». Перша і друга закладки мають кнопки «Відкрити проект» та «Створити проект». Друга закладка буде більш корисною, якщо ви відкриваєте середовище для продовження роботи над проектом.

Режим редагування використовується для роботами з текстами програм під час їх створення та налагодження. Далі цей режим буде розглянуто докладніше.

Режим дизайну використовується для створення графічних інтерфейсів користувача (GUI). Робота у цьому режимі буде розглядатися у наступному семестрі.

Режим роботи з проектом використовується для налаштування параметрів проекту, якщо він відкритий. Зокрема, у цьому режимі можна перевірити таблицю кодування, що використовується у проекті. Для цього слід відкрити закладку «Налаштування редактора» на панелі режиму «Проекти».

Режим налагодження використовується для роботами з кодом програми під час налагодження.

Режим довідки надає доступ до довідкових матеріалів. На лихо деяких студентів, довідка англомовна. So, they have to learn English.

1.1.3 Панель доступу до результатів опрацювання проекту

Ця панель розташована у нижній частині вікна і містить декілька кнопок для доступу до панелей різного призначення. Слідкувати за опрацювання проекту можна скориставшись кнопкою «Консоль збірки».

Консоль додатку відкривається автоматично, коли програма починає працювати.

1.1.4 Створення проекту

Для створення проекту слід відкрити панель режиму «Початок» та натиснути кнопку «Створити проект».

Середовище "Qt creator" 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.

Після вибору типу проекту слід дати цьому проекту ім’я та вибрати папку де проект буде зберігатися. Для імен папок та проекту слід використовувати латинський шрифт. Доречно зберігати усі проекти в одній папці та давати проектам зрозумілі імена.

Контролем версій на першому курсі ми займати не будемо, тому далі можна просто вибрати кнопку «Завершити».

1.1.5 Панель редагування програм

Після створення проекту автоматично відкривається панель редактора коду. Ця панель має дві складові. Ліворуч знаходиться панель, що відображає склад проекту. Праворуч розташована панель редагування коду.

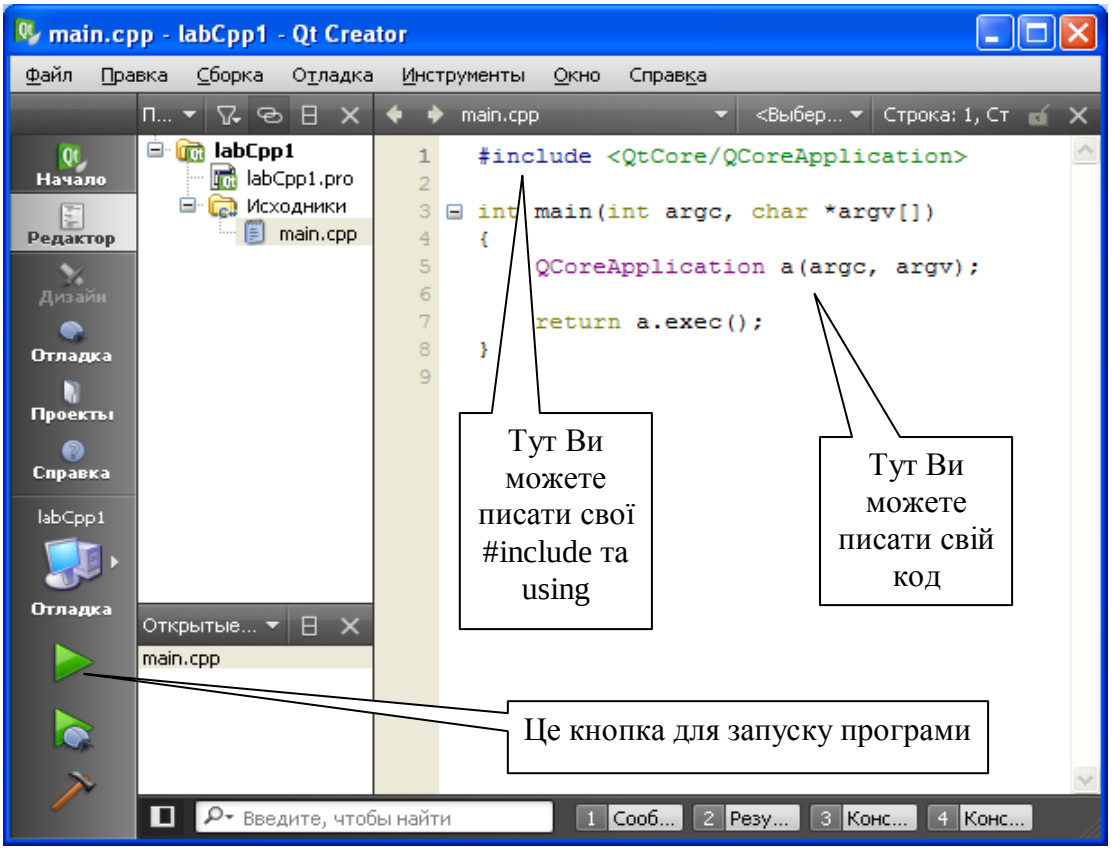


Рисунок 1.2 – Панель редагування програм

Як свідчить ліва панель, проект, який було створено автоматично, розташовано у папці, ім’я якої співпадає з ім’ям проекту. Папка одразу ж містить декілька файлів. Файл із розширенням .pro містить службову інформацію, його можна відкрити і подивитися на його вміст, але зараз нам це не цікаво.

Нас має зацікавити файл main.cpp. Це саме та програма, що має виконуватися. ЇЇ можна запустити на виконання. Для цього у пункті «Зборка» головного меню слід вибрати функцію «Виконати», або натиснути клавіші “Ctrl”+”R”, або натиснути зелений трикутник на панелі ліворуч.

Поки що програма нічого особливого не робить. Просто з’являється вікно консолі і більш нічого.

Але ми можемо розширити можливості програми і одночасно спробувати попрацювати з редактором коду.

Перш за все напишемо ще одну директиву #include такого змісту:

```
#include <iostream>
```

На рисунку 1.2 показано місце, де слід написати цей рядок.

До речі, написавши декілька символів, натисніть клавіші “Ctrl”+”Пробіл”. Як правило, після цього Ви отримаєте можливість вибрати слово цілком.

Директива #include <iostream> означає, що ми підключаємо бібліотеку C++ для потокового вводу-виводу через консоль.

Окрім того слід підключитися до простору імен компілятора C++. Для цього потрібно написати ще один рядок:

```
using namespace std;
```

Тепер ми маємо можливість спілкуватися з програмою через консоль.

Попросимо програму вивести на консоль привітання «Привіт першокурснику!».

Для цього достатньо дописати у файл main.cpp один рядок:

```
cout << “Привіт першокурснику!\n”;
```

Місце, куди потрібно вставити цей код показано на рисунку 1.2.

У написаному рядку cout – це об’єкт C++, що відповідає за виведення інформації на консоль. У подвійних лапках наводиться текст, що підлягає виведенню. Символи «\n» означають, що після виведення тексту слід перейти до нового рядка. А операція << означає, що текст "Привіт першокурснику! \n" буде передано об’єкту cout.

Спробуйте виконати цю програму.

1.1.5.1 Автоматичне вирівнювання тексту програми

Мова C++ не має якихось серйозних обмежень на те, як писати код програми. Однак працювати з програмою буде легше, якщо її стиль буде зручним для сприйняття. Можна рекомендувати такий стиль написання програми:

– кожний оператор має займати окремий рядок;

– відкриваюча і закриваюча фігурні дужки для функції розташовуються кожна в окремому рядку;

– оператори функції розміщуються з відступом від фігурних дужок;

– пробіли не ставляться з обох сторін круглих дужок, що йдуть за круглими дужками.

За допомогою контекстного меню панелі редагування (викликається

правою кнопкою миші) можна вирівняти фрагмент програми (авто відступ), якщо виділити цей фрагмент.

Спробуйте зробити це, попередньо розташувавши рядки нерівно.

1.1.5.2 Коментарі до тексту програми

Коментарі у мовах програмування використовуються для пояснень коду. Компілятор коментарі ігнорує, тому програміст може писати у коментарях що завгодно.

Окрім того, коментарі часто використовують для тимчасового закриття ділянок коду від компілятора.

У мові C++ коментарі позначаються двома похилими рисками.

Ви можете коментувати виділені фрагменти коду за допомогою того ж контекстного меню та відміняти ці коментарі.

Спробуйте зробити це.

1.1.5.3 Повернення у режим редагування

Клавіша Esc забезпечує повернення до редактора із будь якого вікна.

1.1.6 Функція main()

Програма на C або C++ являє собою сукупність різноманітних оголошень та функцій. Функція - це оформлений стандартним чином, логічно завершений фрагмент коду, що має власне ім’я, вирішує деяку локальну задачу і може повертати якийсь результат.

Кожна програма на C та C++ обов’язково повинна мати у своєму складі функцію з назвою main(). Це головна функція програми з якої починається виконання кожної програми. Структуру функції поясняє рисунок 1.3

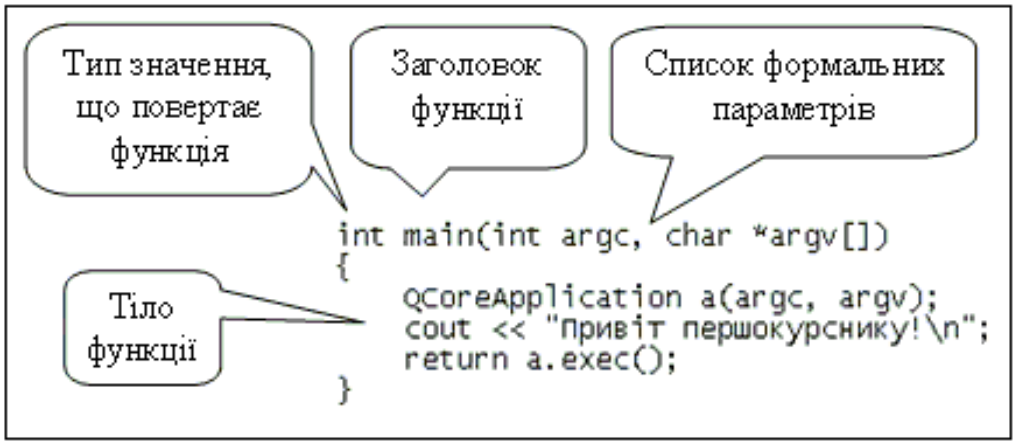


Рисунок 1.3 - Структура функції main()

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

У більшості середовищ розробки очікується, що функція main() має повертати код завершення - якесь ціле число, що повідомляє, яким чином

завершилася програма. Число 0 зазвичай вказує що програма завершилася нормально. Це число повертає оператор return.

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

У найпростішому випадку функція main() може мати і такий вигляд:

```
int main()
{
        …
        return 0;
}
```

У функції main(), що наведена на рисунку 1.3 і створюється разом із новим проектом дещо складніший вигляд. Тіло містить описи та оператори, що забезпечують відкриття вікна консолі.

Ви можете використовувати і спрощений вигляд функції, вилучивши усе те, що було створено автоматично, але у цьому випадку доведеться вирішувати проблему, як зберегти на екрані вікно консолі після завершення роботи функції main().

У прикладах, що будуть наводитися далі, ми будемо використовувати спрощений вигляд функції main(), але це робиться тільки з метою скорочення тексту.

1.1.7 Ще один приклад простої програми

На завершення лабораторної роботи ми трохи розширимо створену програму (рисунок 1.4).

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <cmath>
using namespace std;
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    cout << "Привіт першокурснику!\n";
    cout << "Qt вміє рахувати:\n"
         << "2+3 = " << 2+3 << "\n";
    cout << "Qt знає тригонометрію:.\n"
         << "sin(0.5) = " << sin(0.5) << "\n";
    cout << "Qt вміє добувати корені:.\n"
         << "корінь квадратний із 2 дорівнює "
         << sqrt(2) << "\n";
    return a.exec();
}
```

Рисунок 1.3 – Приклад простої програми

Зверніть увагу на те, що у програму добавлено директиву #include <cmath>, яка забезпечила доступ до математичних функцій.

Проаналізуйте варіанти використання об’єкту cout.

1.1.8 Збереження проекту

Для збереження проекту достатньо вибрати функцію «Зберегти все» із головного меню. Усі файли будуть збережені у папці, яку ви вибрали створюючи проект. Окрім папки з проектом створюється ще одна папка, ім’я якої окрім імені проекту включає ще слова «-build-desktop».

Якщо проект треба буде розгорнути на іншому комп’ютері, то можна скопіювати обидві папки і розгорнути ці папки там де потрібно.

Можна також зберегти тільки файл main.cpp, а на іншому комп’ютері створити новий проект і у ньому замінити зміст файлу main.cpp.

1.1.9 Етапи виконання програми

Безпосередньому виконанню програми, що написана на мові С++, передує декілька етапів перетворення її програмного коду. Основними з цих етапів є: препроцесорне опрацювання, компіляція програмних елементів, компонування об’єктних кодів складових частин програм у єдиний виконавчий код у вигляді файлу .exe.

Докладніше про ці етапи Ви маєте прочитати у підручнику [1] , стр.27.

1.2 ВИМОГИ ДО ЗВІТУ

- Назва роботи.
- Мета роботи.
- Короткі теоретичні відомості.
- Тексти функцій програми з коментарями.
- Результати тестування проекту у вигляді копій консолі.
- Висновки.

1.3 КОНТРОЛЬНІ ПИТАННЯ

- Структура програми на С.
- Структура функції main().
- Найпростіший варіант написання функції main().
- Призначення директив #include <cmath> та #include <iostream>
- Як користуватися об’єктом cout.
- Робота з вікном редактора коду.
- Як зберегти проект.
- Як перенести проект на інший комп’ютер.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2006. – 432 с.

2 ЛАБОРАТОРНА РОБОТА № 2. ТИПИ ДАНИХ ТА РОЗРАХУНКИ ЗА ФОРМУЛАМИ

Мета роботи:
- Познайомитися з поняттям тип даних та типами даних мови С, С++.
- Познайомитися з поняттям змінна
- Навчитися записувати арифметичні вирази.
- Створити додаток, що забезпечує розрахунок за формулою.

2.1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1.1 Змінні і константи

Програмні проекти зазвичай розробляються для обробки деяких даних з метою отримання інших даних. Наприклад, у проекті «Стипендія» вихідними даними є оцінки студента і його статус, а вихідними - розмір стипендії та розмір прибуткового податку. Для представлення даних використовують поняття «змінна» і «константа». Різниця між змінною і константою полягає тільки в тому, що константа не може змінювати своє значення під час роботи програми. В якості імені змінної можна використовувати послідовність з літер латинського алфавіту, цифр та символу підкреслення. Першим символом в імені змінної повинна бути буква. Пробіл в імені змінної використовувати не можна. Великі і маленькі літери розрізняються.

2.1.2 Типи даних

Будь-які дані в пам'яті комп'ютера зберігаються як послідовності нулів та одиниць, але для того, щоб визначити, що означає така послідовність, необхідно знати, до якого типу вона відноситься. Тому у мовах С, С++ тип кожної змінної повинен бути обов'язково зазначений при оголошенні.

Наприклад, нехай у пам'яті комп'ютера записана послідовність нулів і одиниць 0100 0010 0100 0011 0100 0100 0000 0000. Якщо розглядати її як ціле число, то це буде 1111704576, а якщо припустити, що це рядок символів, то отримаємо «BCD».

З наведеного прикладу випливає, що тип даних визначає спосіб кодування інформації.

Окрім того, для кожного типу визначено свій набір допустимих операцій над даними та спосіб їх виконання.

Так, наприклад, якщо для розглянутої послідовності виконати операцію «скласти сама із собою», то ця операція буде виконуватися за різними правилами.

Числа будуть складатися за правилами арифметики, і отриманий результат буде виглядати так: 1000 0100 1000 0110 1000 1000 0000 0000 (2223409152).

Рядки ж будуть "склеюватися" і результат буде таким: 0100 0010 0100 0011 0100 0100 0100 0010 0100 0011 0100 0100 0000 0000 («BCDBCD»).

Мова С++ надає програмісту можливість використовувати декілька різновидів типів.

Ці типи можна поділити на такі:

– скалярні, які поділяються на арифметичні, переліки та вказівники;
– тип функція;
– агреговані, що складаються за певними правилами із даних скалярних типів та, можливо, функцій.

Поки що ми розглянемо тільки скалярні арифметичні типи даних. Коротку інформацію про інші типи можна знайти у підручнику [4], стр.31-37.

2.1.3 Арифметичні типи даних

До арифметичного типу даних у С++ відносять дані, до яких можна застосовувати усі арифметичні операції.

Ці типи поділяють на основні та модифіковані.

2.1.3.1 Основні арифметичні типи

Основними типами є такі:
char – тип для символів;
int – тип для цілих чисел;
float – тип для дійсних чисел;
double – дійсні числа подвійної точності;
bool – логічний тип, що може приймати значення 0, або 1.

У читача може створитися враження, що тут помилка. Дійсно дивно, що типи char та bool віднесені до арифметичного типу. Але помилки тут нема. Вираз 'Z'*true є допустимим у С і його можна обчислити. Результатом буде число 90, бо true перетворюється у 1, а код символу ’Z’ дорівнює 90. Тобто логічну змінну і символ можна розглядати як цілі числа.

Слід також прийняти до уваги, що тип bool з’явився тільки у мові С++. А в мові С число 0 розглядалося як true, а будь яке інше число розглядалося як false.

2.1.3.2 Модифіковані арифметичні типи

Модифіковані типи отримують за допомогою модифікаторів. Значення модифікаторів та типи, до яких їх можна застосовувати показані у таблиці 2.1.

Таблиця 2.1 – Модифікатори для типів мови С++

Модифікатор	Значення	Застосовують до типів
signed	із знаком	char, int
unsigned	без знаку	char, int
long	довгий	int, long int, double
short	короткий	int

Як бачимо, використання модифікаторів обмежено, за виключенням типу int, до якого можна застосувати будь який модифікатор. Та на практиці область використання ще вужча. Модифікатор signed практичного сенсу не має, бо за

принципом замовчування усі типи мають знак. Для знаку виділяється старший біт коду числа. Якщо значення цього біту нуль, то число додатне, якщо одиниця - то від'ємне.

Модифікатор short зменшує удвічі довжину типу int, а модифікатор long збільшує удвічі довжину основного типу. Модифікатор long для типу int можна використовувати двічі. У цьому разі розмір типу int збільшується у чотири рази. Ці модифікатори можна комбінувати з модифікатором unsigned.

У Qt можливі і скорочення, що визначені у заголовному файлі <qtglobal>. Наприклад, замість типу unsigned int можна писати uint. Перелік скорочених назв типів наведено у таблиці 2.2. У цій же таблиці наведено перелік відповідних типів бібліотеки QT, які також визначені у заголовному файлі <qtglobal>.

2.1.3.3 Граничні значення даних цілочислових типів даних

Граничні значення різних типів даних можуть залежати від програмного середовища та типу комп'ютера. Для того щоб зменшити цю залежність, у заголовному файлі <climits> записано набір констант, значення яких дорівнюють мінімальним та максимальним значенням кожного цілочислового типу. Мінімальне значення усіх без знакових констант дорівнює 0.

Таблиця 2.2 – Скорочені назви та граничні константи для типів C++

Повна назва типу	Скорочена назва	Тип Qt	Граничні константи
char	char	qint8	CHAR_MIN
char	char	qint8	CHAR_MAX
unsigned char	uchar	quint8	UCHAR _MAX
short	short	qint16	SHRT_MIN
short	short	qint16	SHRT_MAX
unsigned short	ushort	quint16	USHRT_MAX
int	int	qint32	INT_MIN
int	int	qint32	INT_MAX
unsigned int	uint	quint32	UINT_MAX
long int	long	qint64	LONG_MIN
long int	long	qint64	LONG_MAX
unsigned long int	ulong	quint64	ULONG_MAX
long long int	long long	qlonglong	LLONG_MIN
long long int	long long	qlonglong	LLONG_MAX
unsigned long long int	?	qulonglong	ULONG_LONG_MAX

2.1.3.4 Переліки

Переліки – це типи, яким відповідають набори цілочислових констант, кожна з яких має унікальне ім'я.

Оголошення переліку має такий вигляд:

```
enum <ім'я переліку> {<список іменованих константз>};
```

Наприклад, якщо ми захочемо перефарбувати вікно консолі, або змінити колір тексту, що виводиться на консоль, нам потрібно буде оперувати з поняттям колір. Кожний колір для консолі кодується цілим числом. Нулю відповідає чорний, одиниці – синій і т.д. Але пам'ятати номери кольорів не дуже зручно. Зручніше користуватися іменами з такого переліку:

```
enum consoleColors{
  BLACK, BLUE, GREN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY,  LIGHTBLUE,
  LIGHTGREN, LIGHTCYAN, LIGHTRED,  LIGHTMAGENTA, YELLOW, WHITE
};
```

Стандартно перший елементу переліку має значення 0, а кожний наступний має значення на 1 більше, ніж попередній.

У разі потреби елементам можна присвоювати інші значення.

2.1.3.5 Цілочислові константи

Цілочислові константи можна записувати у трьох формах: десятковій, вісімковій та шістнадцятковій.

Десяткова константа являє собою послідовність десяткових цифр, перед якою може бути знак + або -. Наприклад, 123, -234, +456.

Вісімкова константа завжди починається з 0 і може мати у своєму складі тільки цифри від 0 до 7. Наприклад, 015, 0777. Зверніть увагу на те, що константи 125 і 0125 мають різне значення. Десяткове значення константи 0125 дорівнює 85 (1*64+2*8+5*1) .

Шістнадцяткові константи позначаються префіксом 0x. До їх складу, окрім десяткових цифр можуть також входити шістнадцяткові цифри (A, B, C, D, E, F). Наприклад, числу 0x12A відповідає десяткове число 298 (1*256+2*16+1*10).

Шістнадцяткові константи дуже зручно використовувати для представлення двійкових кодів, бо кожній шістнадцятковій цифрі відповідає 4 двійкових (тетрада). Так число 0x815F у двійковій формі буде виглядати так: 1000 0001 0101 1111.

Тип константи встановлюється компілятором за її значенням. Для не занадто великих констант встановлюється тип int, а якщо ця константа вісімкова або шістнадцяткова то тип unsigned int. Але є можливість і примусово встановити тип константи. Для цього до константи долучають кінцеві символи U (unsigned) та L (long). Наприклад, константа 1UL, незважаючи на те, що вона має маленьке значення отримає тип unsigned long.

2.1.3.6 Символьні константи

Символьні константи поділяються на графічні символи , керуючі слеш-символи та ескейп-послідовності. У будь якому варіанті символьні константи

розташовують між одинарними лапками. Наведемо приклади символьних констант: ‘a’ (літера a), ‘\a’ (звуковий сигнал), ‘\x0a’ (перехід на новий рядок).

Більш докладно із спеціальними керуючими символами ви можете ознайомитися у підручнику [4] на сторінках 12-16 та 34-35.

2.1.3.7 Константи дійсних типів

У пам’яті комп’ютера дані цього типу зберігаються у формі з плаваючою крапкою. При такому записі число представляється у вигляді мантиси і порядку. Стандартно мантиса у двійковому коді починається з 1, а десяткова має значення більше або рівне 1, але менше 10. Порядок може бути позитивним чи негативним і показує, на скільки порядків реальне число менше або більше мантиси. Перевага такої форми запису полягає в тому, що не доводиться писати незначущі нулі. Довжина запису залежить тільки від кількості значущих цифр і не залежить від значення числа. Наприклад, числа 123450000 і 0.0000000012345 будуть представлені як 1.2345e +8 і 1.2345e-9. Це зручно як для запису чисел на папері, так і для зберігання чисел в пам'яті комп'ютера. Та у програмах дотримуватися цього стандарту не обов’язково. Інколи зручніше записувати дійсні константи у звичайній формі, з фіксованою крапкою.

Дійсний тип даних дозволяє оперувати з дробовими десятковими числами в дуже широкому діапазоні і з високою точністю, тому дані цього типу найбільш часто використовуються для інженерних розрахунків. З граничними параметрами дійсних типів можна ознайомитися у підручнику[1] на сторінці 37.

За правилом замочування константи зберігаються у пам’яті як числа типу double. Але є можливість і примусово встановити тип константи. Для цього до константи долучають кінцеві символи F (float) або L (long double).

2.1.4 Описи змінних

Усі змінні у програмі на мові С мають бути описані, (оголошені). Оголошення змінної - це інструкція, в якій зазначається ім'я змінної та її тип.

У простому випадку інструкція оголошення змінної виглядає так:

```
<тип> <имя>;
```

де: <ім'я> - ім'я змінної;

<тип> - ім'я типу для оголошуваної змінної.

В одній інструкції можна оголошувати декілька змінних одного типу. У цьому випадку імена змінних розділяються комами.

У розділі може бути і декілька інструкцій оголошення, наприклад:

```
int i, j, k ;
float x;
```

Описуючи змінну, можна одразу надати їй значення. Це зветься ініціалізацією. Інструкція ініціалізації змінної виглядає так:

```
<тип> <имя> = < вираз ініціалізації >;
```

У якості виразу ініціалізації може бути константа, змінна, яка вже має значення, або вираз.

Якщо значення, що прийняла змінна після ініціалізації на повинно змінюватися, то для таких змінних використовують кваліфікатор const. Використання такого кваліфікатор перетворює змінну у константу. Нижче наведено приклад оголошення константи 2π.

```
const double twoPi=2*3.1425926;
```

2.1.4.1 Макроконстанти

У мові C++ існує можливість задавати константи за допомогою макропідстановок або макросів, які визначаються за допомогою директиви #define препроцесора, яка виглядає так:

```
#define <ім'я макросу> < текст заміни>
```

Ця директива виконує заміну *ім'я макросу* на *текст заміни* в усьому тексті програми, починаючи з наступного за директивою рядка.

За звичай для імен макросів використовують заголовні літери, щоб відрізняти їх від звичайних імен.

Наприклад:

```
#define PI    3.1415927
#define TWOPI   (2*PI)
```

2.1.5 Операція розміру sizeof

Операція sizeof повертає обсяг пам'яті, яку займає або потребує операнд цієї операції, тобто розмір операнда. В ролі операнда може бути змінна, вираз або назва типу. Розмір визначається в байтах. Байт – це основна одиниця виміру обсягу пам'яті в інформатиці. Один байт дорівнює 8 біт. Біт це найменша одиниця виміру обсягу інформації, що відповідає одному розряду двійкового коду і може приймати значення нуль або один.

Операція sizeof має декілька стандартних форм:

```
sizeof (тип)
sizeof  імя змінної
sizeof  (вираз)
```

Розмір операндів різних типів у мові С не є визначеним і залежить від системи програмування та типу комп'ютера.

2.1.6 Арифметичні операції

Перелік арифметичних операцій C++ наведено у таблиці 2.3. Усі наведені операції, окрім операції %, виконуються над даними усіх числових типів. Операція % має сенс тільки для цілих чисел.

Таблиця 2.3- Арифметичні операції C++

- унарний	Зміна знаку операнда
*	Множення
/	Ділення
%	Остача від цілочислового ділення
+	Додавання
-	Віднімання

Слід також мати на увазі, що операція ділення для цілих чисел повертає також ціле число. Для того, щоб результат ділення був дійсним числом треба, щоб хоча б один операнд був також дійсним числом. Наприклад, 12 / 5 буде 2, але 12.0 / 5 буде 2.4.

2.1.7 Операції присвоєння

2.1.7.1 Проста операція присвоєння

Операція присвоєння є однією із основних операцій у будь якій мові програмування. У мові C++ вона виглядає так:

```
<змінна> = <вираз>;
```

де: <*змінна*> - ім'я змінної, значення якої змінюється в результаті виконання інструкції присвоювання;

= знак операції присвоювання.

<*вираз*> – вираз, значення якого присвоюється змінній, ім'я якої вказане ліворуч від символу присвоювання.

Присвоювання виконується наступним чином:

– спочатку обчислюється значення виразу, який знаходиться праворуч від знаку операції присвоювання;

– потім отримане значення записується у змінну, ім'я якої стоїть ліворуч від символу присвоювання.

Операція присвоювання вважається вірною, якщо тип значення, що повертає вираз, відповідає або може бути приведений до типу змінної, яка отримує це значення. Наприклад, змінній типу double можна присвоїти значення виразу, тип якого float або int.

Особливість операції присвоювання у мові С полягає у тому, що ця операція **повертає результат**, що дорівнює значенню виразу у правій частині. Тому цю операцію можна використовувати у виразах. Слід тільки мати на увазі, що ця операція має низький пріоритет. Тому присвоєння у виразах слід брати в дужки. Наприклад, вираз a + (b = c+d) є допустимим у мові С.

2.1.7.2 Комбіновані присвоєння

Окрім простого оператора присвоєння у мові С завжди були ще і так звані комбіновані оператори присвоєння. Згодом вони з’явилися і у інших мовах.

Ці оператори використовують у тих випадках, коли ліворуч і праворуч від знака операції присвоєння знаходиться той самий операнд.

Наприклад, замість присвоєння number = number + d можна записати number += d.

Таке присвоєння можна використовувати у комбінації з будь якою арифметичною операцією та багатьма іншими, наприклад, *=, /=, %=, тощо.

Комбіновані присвоєння скорочують запис і вважається, що вони роблять запис більш наочним і скорочують процес обчислення результату. Стиль написання програм на мові С схвалює використання таких присвоєнь.

2.1.7.3 Унарні присвоєння

Ці присвоєння є окремим випадком комбінованого присвоєння. Вони використовуються тоді, коли до змінної треба додати одиницю або відняти її. Відповідно до цього маємо операції інкременту та декременту.

Запис операції унарного присвоєння ще простіший ніж комбінованого. Для того щоб, наприклад, збільшити змінну number на одиницю, можна написати number++, або ++ number. Так само можна і зменшувати значення на одиницю: number--, або --number.

Операції ++ та -- називають префіксними, якщо знаки цих операцій записуються перед змінною. Якщо ж знаки операції записуються після змінної то операції називають постфіксними.

Різниця між префіксними та постфіксними операціями проявляється у тих випадках, коли ці операції використовуються у виразах разом з іншими операціями. Справа у тому, що префіксні операції мають найвищий пріоритет, а постфіксні – найнижчий. Хай, наприклад, змінна x має значення 5. Тоді значення виразу (3+ ++x) буде дорівнювати 9, а значення виразу (3+x++) буде дорівнювати 8, хоча «x» у обох випадках отримає значення 6.

2.1.8 Вирази

Вираз - це послідовність операндів, об’єднаних знаками операцій та круглими дужками. У мовах високого рівня вираз мало відрізняється від формули. Основна відмінність полягає в тому, що вираз записується в один рядок. У таблиці 2.4 наведені приклади запису деяких виразів

Таблиця 2.4 Приклади запису виразів

Формула	Вираз
$\frac{a+b}{(a-b)x}$	(a+b)/(a-b)/x
$\sqrt{\left(\sin^2 x+b\right)}$	sqrt(pow (sin(x),2)+b)
$e^{2.5x}$	exp(2.5*x)
$a^{1.5+b}$	pow(a, 1.5+b)

Як вже було сказано, вираз складається з операндів і знаків операцій. В якості операндів виразу можна використовувати: змінну, константу, функцію

або інший вираз. Знаки операцій знаходяться між операндами і позначають дії, які виконуються над операндами.

У найпростішому випадку вираз може являти собою константу або змінну.

При обчисленні значень виразів слід враховувати, що операції мають різний пріоритет. Операції множення і ділення, наприклад, мають вищий пріоритет, ніж операції додавання і віднімання.

При обчисленні значення виразу в першу чергу виконуються операції з більш високим пріоритетом. Якщо пріоритет операцій у виразі або його частини однаковий, то операції, зазвичай, виконуються зліва направо, але є виключення.

Якщо потрібно змінити порядок виконання операцій у виразі, слід використовувати дужки. Вираз у дужках трактується як один операнд. Це означає, що операції над операндами в дужках будуть виконуватися в звичайному порядку, але раніше, ніж операції над операндами, розташованими за дужками. При записі виразів, що містять дужки, слід дотримуватися парності дужок, тобто скільки дужок відкрито стільки має бути і закрито.

2.1.9 Пріоритети операцій у C++

У таблиці 2.5 наведено приорітети операцій C++. Чим менший номер пріоритету, тим вищий пріоритет. Деякі операції, можливо, вам незнайомі, та колись ви про них дізнаєтесь.

Таблиця 2.5 – Пріоритети операцій C++

Операції	Пріоритет
Префіксні ++ -- ( ) [ ] . ->	1
! ~ +a -a (type) sizeof *a &a	2
* / % (арифметичні)	3
+ - (арифметичні)	4
<< >> (зсуви ліворуч і праворуч)	5
< > <= >= (порівняння)	6
= = != (порівняння)	7
& ^ | (порозрядні логічні операції)	8,9,10
&& || (логічні операції)	11,12
? : (теренарна операція)	13
= += -= *= /= %= &= ^= |= >>= (присвоєння)	14
Постфіксні ++ -- , (операція кома)	15

2.1.10 Узгодження типів

У мові C++ є допустимою ситуація, коли операнди операції мають різний тип. У цьому випадку компілятор перевіряє сумісність операндів і встановлює для них спільний тип. Цей же тип і буде типом результату операції. Таке перетворення виконується компілятором автоматично, тому і перетворення звуться автоматичними.

Є дві схеми автоматичного узгодження типів: арифметичні перетворення і перетворення під час присвоєнь.

2.1.10.1 Арифметичні перетворення

Ці перетворення виконуються у арифметичних та порозрядних операціях та операціях порівняння.

Під час цих перетворень перш за все виконується так званий integral promotion: типи bool, char і short перетворюються у int .

bool, char, short → int

Зокрема, перетворення bool → int перетворює true в 1, a false у 0.

Далі використовується принцип ”найбільшого операнду”, відповідно до ряду:

long double, double, float, usigned long, long, unsigned int, int

Тобто якщо, наприклад, у виразі є операнди типу float, int та double то усі операнди будуть приведені до типу double.

2.1.10.2 Перетворення типів в операціях присвоєння

У результаті операції присвоєння результат отриманий у правій частині приводиться до типу лівої частини. При цьому:

– якщо тип змінної ліворуч знаку операції присвоєння вище типу результату обчислення правої частини, то результат обчислення перетворюється за принципом ”найбільшого операнду”;

– якщо тип змінної ліворуч знаку операції присвоєння нижче типу результату обчислення правої частини, то результат обчислення обтинається. Це полягає у тому, що може бути зменшено число розрядів мантиси (double->float), або відкинуто дробову частину (float->int). Але внаслідок таких перетворень можливо і спотворення результату. Наприклад, перетворення long -> int може призвести до втрати старших біт результату.

2.1.10.3 Явне перетворення типів

Мова C++ дозволяє і явне перетворення даних одного типу у інший. Для цього можна використовувати такі конструкції:

(*тип*) *вираз*

тип (*вираз*)

Останній варіант з’явився у C++ і схожий на виклик функції. Але на відміну від виклику функції тут перед круглими дужками рекомендують ставити пробіл.

2.1.11 Бібліотека математичних функцій cmath

Бібліотека математичних функцій cmath надає програмісту можливість використовувати різні функції для обробки даних.

Звернення до цих функцій використовуються у виразах і розглядаються як операнди. Для виклику функції необхідно записати її ім'я і далі в дужках, через кому, перерахувати необхідні параметри. Наприклад, щоб обчислити квадратний корінь із змінної n, достатньо записати sqrt (n).

При роботі з функціями слід враховувати тип значення, що повертається і типи параметрів. В одних випадках виконується автоматичне приведення типів даних, в інших виникає невідповідність і компілятор виводить повідомлення про помилку.

У таблиці 2.6 наведено деякі функції бібліотеки cmath.

Таблиця 2.6 Функції бібліотеки cmath

Приклад звернення до функції	Що обчислює функція
fabs(вираз)	Модуль числа
sqrt(вираз)	Квадратний корінь
pow(вираз, ступінь)	Піднесення до ступеню
exp(вираз для степеню числа e)	Експонента
log(вираз)	Натуральний логарифм
log10(вираз)	Логарифм за основою 10
sin(вираз)	Синус
cos(вираз)	Косинус
tan(вираз)	Тангенс
asin(вираз)	Арксинус
acos(вираз)	Арккосинус
atan(вираз)	Арктангенс
sinh(вираз)	Гіперболічний синус
cosh(вираз)	Гіперболічний косинус
tanh(вираз)	Гіперболічний тангенс
celi(вираз)	Округлення до найближчого більшого
floor(вираз)	Округлення до найближчого меншого
fmod(вираз ділене, вираз дільник)	Залишок від ділення з плаваючою точкою

2.1.12 Використання об’єкту cin для введення даних

У мові C++ введення даних з консолі розглядається як потік символів з клавіатури у програму. Об’єкт **cin** і представляє цей потік у програмі.

Операція >> бере інформацію з цього потоку і перетворює її відповідно до типу змінної, що отримує цю інформацію.

Оператори введення інформації з консолі можуть виглядати так:

```
cin >> x;
cin>>a>>b
      >>c;
```

У першому випадку об’єкт забезпечує введення даних у змінну x.

У другому – до змінних a,b,c.

Об’єкт починає працювати після того, як користувач введе дані з клавіатури і натисне клавішу “Enter”. Вхідні дані розділяють пробілами, табуляцією або переходом до нового рядка.

2.1.12.1 Форматування арифметичних даних під час виведення

Точність представлення чисел дійсного типу дуже висока. Результат обчислень може мати велику кількість значущих цифр. Припустимо, що в результаті обробки результатів експерименту на лабораторній роботі з фізики, ми отримали значення деякої величини рівне 9.85432, хоча вихідні дані для розрахунку мали точність значно меншу. Такий результат краще форматувати перед виведенням, округливши до необхідної кількості цифр після крапки

Об'єкт cout можна налаштовувати для форматування за допомогою його методів setf (), precision (),.width(), fill () та інші.

Докладнішу інформацію про форматування арифметичних даних під час виведення за допомогою cout можна знайти у [2], стр.911- 927, а також на сайті:

http://www.c-cpp.ru/books/formatirovanie-s-pomoshchyu-funkciy-chlenov-ios

Тут ми наведемо як приклад тільки фрагмент програми, який ви можете додати до функції main() і подивитися результат.

```
cout << "Науковий формат і знак + :\n";
cout.setf (ios::showpos);
cout.setf (ios::scientific);
cout <<123 << " " << 123.23<<" " << 123 <<"\n";
cout<<"10 значущих цифр після коми:\n";
cout.precision (10);
cout<<1/3.0<<" " <<0.1<<"\n";
cout<<"Ширина поля 10 знаків:\n";
cout.width(10);
cout << 123 <<" "<< 123.23<<" "<<123 << "\n";
cout<<"Використання символів заповнювачів:\n";
cout.fill ('#');
cout.width(10);
cout <<123<< " " << 123.3<< " "<<123<< "\n";
```

2.1.13 Зміна кольорів консолі та керування курсором

Ці питання виходять за межі нашої програми, але якщо у когось є бажання цим займатися, то можна у проект додати нижченаведений фрагмент програми, або створити заголовний файл з таким змістом і підключати його директивою #include до будь якого проекту.

```
#define WIDTH 80 //ширина консолі у символах
#define HEIGHT 25 //висота консолі у символах
```

```
//перелік кольорів консолі
enum consoleColors{
   BLACK, BLUE, GREN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY,
   DARKGRAY, LIGHTBLUE,  LIGHTGREN, LIGHTCYAN, LIGHTRED,
   LIGHTMAGENTA,YELLOW, WHITE
};
/* Пояснення:
   HANDLE (дескриптор, опис чогось) -  число, що задає номер якого ресурсу
   (тут ресурс - це стандартний пристрій виведення на консоль).
   Оскільки користувач працює з дескриптором, а не з номером,
   то номер може змінюватись, але це не позначиться на роботі з пристроєм.
 */
// Отримуємо номер консолі
HANDLE hdlConsole = GetStdHandle(STD_OUTPUT_HANDLE);
//функція зміни кольору
void setColors(int textColor, int backColor)
{
   backColor <<= 4;  //backColor має перейти у старший напівбайт
   WORD color = backColor | textColor; //Об'єднуємо кольори
   SetConsoleTextAttribute(hdlConsole, color); //Передаємо кольори консолі
}
//функція визначення положення курсору
void getCursorPos(int &x, int &y){
   CONSOLE_SCREEN_BUFFER_INFO bi;
   GetConsoleScreenBufferInfo(hdlConsole, &bi);
   x=bi.dwCursorPosition.X;
   y=bi.dwCursorPosition.Y;
}
//функція зміни положення курсору
void gotoXY(int X, int Y)
{
   COORD coord = { X, Y };
   SetConsoleCursorPosition(hdlConsole, coord);
}
```

Якщо додати цей код до програми, то після цього можна змінювати кольори і керувати положенням курсору.

Ось приклад:

```
#include <windows.h>
#include <iostream>
using namespace std;
//ТУТ СЛІД ДОДАТИ ВИЩЕНАВЕДЕНИЙ ТЕКСТ, АБО #INCLUDE
int main(int argc, char *argv[])
{
  QCoreApplication a(argc, argv);
   setColors(YELLOW, BLUE);
   system("cls"); // очищаємо екран кольором BLUE
   //Заголовок вікна консолі
   SetConsoleTitleA("Зміна кольорів консолі та керування курсором на екрані");
   string s="Тут центр екрану консолі";
   int x =  (WIDTH - s.length())/2;
   int y = HEIGHT/2-1;
   gotoXY(x,y);
   cout<<s<<"\n";
  return a.exec();
}
```

2.2 ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

Створити програму, яка б видавала на консоль розмір основних типів даних та їх граничних значень.

Створити програму, яка б видавала результат виконання усіх арифметичних операцій для введеної пари чисел.

Створити програму, яка б видавала результати обчислення кожної, з наведених у таблиці 2.6 функцій. У програмі використати різні види форматування результатів.

2.3 ВИМОГИ ДО ЗВІТУ

- Назва роботи.
- Мета роботи.
- Короткі теоретичні відомості.
- Тексти програм з коментарями.

– Результати виконання програм у вигляді копій консолі.
– Висновки.

2.4 КОНТРОЛЬНІ ПИТАННЯ

– Змінні і константи.
– Тип даних.
– Арифметичні типи даних.
– Описи змінних.
– Операція розміру sizeof.
– Арифметичні операції.
– Операції присвоєння.
– Вирази.
– Пріоритети операцій у С++.
– Узгодження типів.
– Бібліотека математичних функцій cmath.
– Використання об’єкту cin для введення даних.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программированя С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с
2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. –1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вільямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

3 ЛАБОРАТОРНА РОБОТА № 3. ФУНКЦІЇ

Мета роботи:
- Познайомитися з поняттям функції у мові С, С++.
- Познайомитися з поняттям прототип функції.
- Познайомитися із способами передачі параметрів у функції.
- Познайомитися із макросами з параметрами.
- Створити програму з використанням функцій.

3.1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Можна сказати, що функції є основними будівельними блоками програми, написаної на мові С. Функція - це оформлений стандартним чином, логічно завершений фрагмент коду, що має власне ім'я, вирішує деяку локальну задачу і може повертати якийсь результат. Дотепер ми мали справу тільки з функцією main(), або із стандартними функціями, наприклад sin(). Але у процесі створення програми програміст, як правило, створює багато своїх функцій. У цій лабораторній роботі ми познайомимося з функціями більш докладно.

3.1.1 Правила написання функцій

Всі функції мови С мають однакову структуру, що виглядає таким чином:

```
тип_значення_що_повертається ім'я_функції(список_формальних_параметрів)
{
  тіло_функції
}
```

Початкова частина функції, яка складається з типу значення, що повертається, імені функції та списку оголошень формальних параметрів, записаного у круглих дужках, називається заголовком функції або сигнатурою.

Тип значення, що повертається, це будь який можливий у мові С тип. Якщо функція нічого не повертає, записується тип void.

Ім'я функції має задовольняти вимогам мови С до написання ідентифікаторів. Після імені функції обов'язково мають бути круглі дужки, незалежно від наявності формальних параметрів.

Список параметрів складається із оголошень формальних параметрів, відокремлених комами. Оголошуються параметри так само, як і звичайні змінні (тип та ім'я), але кожен з параметрів оголошується окремо, навіть, якщо типи сусідніх параметрів співпадають.

Тіло функції, що знаходиться у фігурних дужках, складається з описів внутрішніх змінних та операторів, що реалізують дії, які повинна виконувати функція. Змінні , що оголошені у тілі функції, називають локальними, тому що вони доступні тільки у межах своєї функції. Вони мають бути оголошені до першого звертання.

Повернення результату роботи функції реалізується через оператор return, який має такий синтаксис:

```
return вираз ;
```

Вираз, що розташований після слова return обчислюється, і його значення є результатом, що повертає функція. Оператор return завершує роботу функції, незалежно від того де він розташований у тілі функції. Якщо алгоритм виконання функції передбачає наявність декількох варіантів завершення, то тіло функції може включати і декілька операторів return.

Якщо функція нічого не повертає, оператор return записується без виразу, або взагалі не використовується.

Нижче, як приклад, наведено опис функції, що повертає суму двох чисел:

```
float  sumTwoNumber (float a,  float b)
{float z;
   z=a+b;
   return z;
}
```

У тілі функції визначено локальну змінну z, яка за межами функції недоступна. Ця змінна приймає результат додавання змінних a та b. Отримане значення функція повертає через оператор return.

Ту саму функцію можна записати і коротше:

```
float  sumTwoNumber (float a,  float b) { return a+b;}
```

3.1.2 Виклик функції

Оператори функції виконуються тільки тоді, коли здійснюється звертання до даної функції. Таке звертання називають викликом функції. Виклик функції виглядає так:

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

Список фактичних параметрів являє собою послідовність виразів, які задають значення формальних параметрів, що перелічені в описі функції. Фактичні параметри обов'язково мають відповідати формальним за кількістю, порядком розташування та типом. **Але не за назвами!**

Виклик функції, що повертає якесь значення, можна використовувати як вираз, а також в якості операнда виразу. Значенням такого операнда буде значення, що повертає функція.

Приклад звернень до функції, опис якої наведено у попередньому пункті, представлено нижче:

У цьому прикладі перше звернення до нашої функції відбувається під час визначення змінної y. Фактичними параметрами у цьому випадку є константи 4.2 та 5.4, а результатом число 9.6. Друге звернення до нашої функції відбувається у виразі, що передається об'єкту cout. Фактичними параметрами у

цьому випадку є змінна x та вираз y/2, а результатом буде число 5.5.

```
…
float  x=2.3;
float  y=sumTwoNumber(4.2, 5.4);
cout << 3+sumToNumber(x, y/3);
```

3.1.3 Прототип функції

Для того, щоб компілятор мав можливість перевірити правильність звертання до функції та використання її результату, опис функції має бути наведено раніше, ніж звертання до неї. Але розташувати таким чином усі функції програми не завжди можливо, а окрім того, і недоречно. Найкраще першою розташовувати функцію main(), далі записувати функції, які послідовно розкривають алгоритм вирішення задачі, а у кінці наводити функції, що деталізують окремі кроки алгоритму.

Для того, щоб у програмах на мові С можна було використовувати довільний порядок розташування функцій використовують прототипи функцій. Прототип функції співпадає із її заголовком і не має тіла, тобто має таку форму:

тип_значення_ що_повертається ім'я_функції(*список_формальних_параметрів*);

Прототип містить усю інформацію, що потрібна компілятору для перевірки правильності застосування функції, і таким чином може замінити для компілятора опис функції, якщо цей прототип розташувати перед викликом функції. Найчастіше прототипи функцій записують перед функцією main(), що є початком програми.

Слід зазначити, що у прототипі можна навіть не наводити імен формальних параметрів, достатньо лише вказати їх типи.

Так, прототип функції, що розглядалася вище як приклад виглядає так:

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

Той же прототип може виглядати і так:

```
float  sumTwoNumber (float,  float);
```

3.1.4 Прототипи бібліотечних функцій

Майже кожна програма на мові С використовує бібліотечні функції. Тексти цих функцій попередньо компілюють, а отримані об'єктні коди записують у бібліотеки. Прототипи цих бібліотечних функцій записують у спеціальних заголовних файлах. Ці файли у мові С мали розширення .h. Але в мові С++ від цього розширення відмовилися. Деякі заголовні файли С були перетворені у заголовні файли С++. Для таких файлів замість розширення .h почали використовувати префікс c. Так, наприклад, файл math.h став файлом cmath.

Для того, щоб підключити до програми заголовні файли бібліотечних функцій, слід застосувати директиву #include. Наприклад:

```
#include <cmath>
```

Слід зазначити, що використання у програмі заголовних файлів C++, за звичай вимагає включення у програму директиви використання простору імен:

```
using namespec std;
```

Тут std – це простір імен стандартних компонентів компілятора C++.

3.1.5 Способи передачі параметрів у функції

Існує два способи передачі параметрів у функції - передача за значенням (by value) і передача через посилання (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 &.

3.1.5.1 Передача параметрів за значенням

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.

Такий спосіб передачі захищає змінні, передані у функцію в якості параметрів, від непередбачуваних змін, оскільки функція працює з копіями. Крім того, такий спосіб дозволяє у якості фактичних параметрів задавати вирази. При передачі параметрів буде обчислено значення виразу і передано у функцію.

Недолік такого способу передачі полягає у тому, що для параметрів, які займають багато пам'яті, наприклад, великі масиви чисел або довгі рядки символів, копії займають багато місця у пам'яті і потребують багато часу для пересилання даних з одного місця пам'яті у інше.

Вище, у прикладі з пункту 3.1.1, параметри до функції передаються по значенню.

3.1.5.2 Передача параметрів через посилання

У разі передачі параметрів через посилання до функції передаються адреси фактичних параметрів. Тому такий спосіб передачі називається ще передачею параметрів за адресою.

При такому способі передачі в якості фактичних параметрів можуть бути тільки змінні. Вираз і навіть окреме число або символ передати через посилання неможливо.

Передача параметрів через посилання заощаджує пам'ять і скорочує час звернення до функції. Однак це має і побічний ефект. Адже функція працює безпосередньо з фактичними параметрами, і будь-яка зміна формального параметру є зміною фактичного параметру. Для запобігання такому ефекту

використовують кваліфікатор const.

Але побічний ефект має і позитивну сторону. Передачу параметрів через посилання можна використовувати для повернення результатів роботи функції через фактичні параметри. Такий спосіб повернення особливо ефективний, коли потрібно повернути декілька параметрів. Адже функція повертає тільки одне значення.

Розглянемо приклад використання передачі параметрів через посилання для повернення результатів роботи функції.

У наведеній нижче функції формальні параметри обмінюються значеннями і фактичні параметри теж обміняються значеннями.

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

3.1.6 Області оголошення та доступу до імен

Область оголошення – це частина програми, в якій можуть здійснюватися оголошення імен змінних, функцій, тощо. Область доступу до імені – це частина програми, у межах якої це ім’я доступно програмісту.

Доступ до імен простягається від точки, в якій це ім’я оголошено, до кінця області оголошення. Незважаючи на те, що оголошення змінних можна розміщувати будь де у межах області оголошення, краще оголошення згрупувати і розмістити на початку області оголошення.

3.1.6.1 Глобальні та локальні змінні

Найширшою областю оголошення є файл, який містить нашу програму. Змінні та константи, що оголошені у файлі, але за межами будь-якої функції називаються глобальними. Глобальні змінні автоматично ініціалізуються нульовим значенням.

Більш вузькою областю оголошення є функція. Імена, що оголошені у функції доступні тільки у межах цієї функції. Змінні, що оголошені у функції називають локальними. Глобальні змінні теж доступні у функціях. Але якщо ім’я локальної змінної у функції співпадає з ім’ям глобальної змінної, то перевага віддається локальній змінній. Тобто локальна змінна перекриває глобальну у межах своєї області доступу.

Найвужчою областю оголошення є блок коду між фігурними дужками. Змінні, що оголошені у блоках теж вважаються локальними.

Локальні змінні автоматично не ініціалізуються. Тому якщо локальна змінна оголошується без ініціалізації (тобто їй не надано початкового значення), то значення цієї змінної може бути будь яким, так зване «сміття».

Глобальні змінні існують поки виконується програма. Пам'ять їм виділяється компілятором і тому їх ще називають статичними.

Локальні змінні створюються тільки після виклику функції та існують тільки до завершення роботи функції, або блоку. Пам'ять для цих змінних

автоматично виділяється, а потім звільнюється у процесі виконання програми. Тому такі змінні ще називають автоматичними.

3.1.6.2 Глобальна чи локальна змінна?

На перший погляд глобальні змінні здаються більш привабливими, оскільки до них мають доступ усі функції. Однак такий легкий доступ до змінних знижує надійність програми.

У більшості випадків слід користуватися локальними змінними і передавати їх іншим функціям тільки через параметри і тільки в разі необхідності.

Однак іноді глобальні змінні доцільно використовувати, наприклад, для зберігання постійних даних, що використовуються декількома функціями. Для запобігання небажаним змінам таких даних можна скористатися ключовим словом const.

3.1.6.3 Специфікатор statsc

Мова С дозволяє, у разі потреби, зберегти значення локальної змінної, яка була обчислена у функції. Це може бути потрібно у тих випадках, коли функція використовує значення, що були обчислені під час попереднього виклику цієї функції. Саме так, наприклад, обчислюються випадкові числа. Кожне наступне підраховується на підставі попереднього. Нижче наведено функцію, що формує цілі випадкові числа за формулою x=a*x+b. На перший погляд тут складається враження, що ніякої випадковості нема, але слід враховувати, що константа «a» дуже велика. Результат множення змінної «x» на таке велике число майже завжди буде перевищувати максимально можливе значення для типу unsigned long, внаслідок чого старші біти результату будуть втрачатися. Таким чином результат стає ніби то випадковим.

```
#include <ctime>
#include <iostream>
using namespace std;
// Функція генерації випадкових чисел
unsigned long amkm() {
   unsigned long a = 0xBL, b = 0x5DEECE66DL;
   static unsigned long x = time(NULL);
   return x=a + b * x;
}
// Тестування функції amkm
int main() {
   m:   cout << amkm() << "\n";
        goto m;
        return (0);
}
```

Константи «a» та «b» задано у 16-річному форматі.

Початкове значення змінної «x» дорівнює значенню поточного часу, що

повертає функція time(). Потім воно змінюється виразом x=a*x+b. Отримане значення зберігається до наступного виклику функції завдяки тому що змінна «x» оголошена із специфікатором **statsc**.

У функції main() функція amkm() безперервно викликається завдяки використанню мітки «m:» та оператора переходу goto.

3.1.7 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
```

У виразі для заміни кожен з параметрів і весь вираз слід брати у дужки.

Як приклад, наведемо макрос для піднесення числа a до ступеня b:

```
#define POW(a,b)  ( exp( (b) * log(a) ) )
```

Звернення до цього макросу у програмі може виглядати так:

```
cout <<  POW( 3 + x , 4.5 ) << "\n"
```

3.2 РЕАЛИЗАЦІЯ ПРОЕКТУ «FUNCTION»

У цій лабораторній роботі слід самостійно створимо проект, який забезпечить розрахунки за деякою формулою. Розрахункову формулу слід вибрати з таблиці 3.1 відповідно до останньої цифри номера залікової книжки.

Таблиця 3.1 – Завдання для проекту

Варіант	Формула
0	$y = a \cdot x^3 - b \cdot \sin(x) + \sqrt{\frac{b \cdot \sin(x)}{a}}$
1	$u = m \cdot tg(k \cdot x) + k + \frac{k \cdot tg(x) + m}{x}$
2	$w = \ln(d \cdot z) - dz + \frac{\ln(c) - z}{c}$
3	$f = a \cdot \sin^4(w \cdot x) - \sqrt[a]{w/x}$
4	$g = a \cdot e^{-b \cdot x} - b \cdot x \cdot e^{-a} \cdot \frac{\sin(c \cdot x)}{c}$

Продовження таблиці 3.1

Варіант	Формула
5	$p = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x+m)}{\sqrt{x \cdot m}}}$
6	$q = \frac{\ln(a \cdot x) + x}{b} + \ln(x - b) + a$
7	$x = \frac{tg(a \cdot t)}{a \cdot t} \sqrt{\lvert b - t \rvert} + \frac{tg(b \cdot t)}{b \cdot t} \sqrt{\lvert a - t \rvert}$
8	$\lambda = a \cdot e^{-x} + \frac{a \cdot x^2 + b \cdot x + 1}{b \cdot x^2 + a \cdot x + 1}$
9	$\mu = \frac{ctg(d \cdot w)}{fw + d} \sqrt{\left\lvert \frac{f + d \cdot w}{ctg(f \cdot w)} \right\rvert}$

Нижче розглядається приклад створення проекту, схожого з тим, який слід розробити. Різниця полягає тільки в розрахунковій формулі і іменах вихідних даних. У цьому проекті створимо програму для розрахунків за формулою 3.1.

$$y = \frac{a \cdot x + \sin^3(a \cdot x)}{\sqrt[b]{a \cdot x/(1+x)}} \tag{3.1}$$

Для обчислень за формулою створимо декілька варіантів функції, які будемо викликати з функції main(). Окрім того будемо використовувати макрос, наведений у попередньому пункті, для піднесення синусу до ступеня 3 та добування кореню ступеня b. Для спрощення перевірки отриманих результатів попередньо у функції будемо обчислювати вираз *ax*, а також числівник і знаменник.

3.2.1 Підключення допоміжних файлів

На рисунку 3.1 наведено перелік файлів, які необхідно включити до програми, щоб забезпечити її нормальне функціонування.

```
#include <iostream>
#include <cmath>

using namespace std;
```

Рисунок 3.1 – Директиви #include програмного файлу

Файл iostream забезпечує реалізацію обміну інформацією з консоллю.
Файл cmath забезпечує доступ до математичних функцій.

3.2.2 Створення макросу з параметрами

Для піднесення синусу до 3-го ступеня та добутку кореня створимо макрос з параметрами, представлений на рисунку 3.2.

```
#define POW( a, b ) ( exp ( (b) * log(a) ) )
```

Рисунок 3.2 – Визначення макросу з параметрами

3.2.3 Створення прототипів функцій

На рисунку 3.3 наведено перелік функцій, які будуть використовуватися у файлі програми. У програмі ми створимо три варіанти функції для обчислення за формулою 3.1.

```
float f1 (float a, float b, float x);
void f2 (float a, float b, float x, float &y);
void f3 (float a, float b, float x, float &y);
```

Рисунок 3.3 – Прототипи функцій програмного файлу

Функція f1() приймає три параметри і повертає значення, обчислене за формулою 3.1.

Функція f2() на відміну від попередньої нічого не повертає, але у якості четвертого параметру приймає посилання на змінну y, до якої і передає результат обчислення за формулою 3.1.

Функція f3() виконує те саме, що і функція f2(), але окрім того виводить на консоль проміжні значення розрахунків, що дозволяє простіше перевірити результати обчислень.

3.2.4 Функція main() програмного файлу

Функція main() програмного файлу забезпечує введення вхідних даних для розрахунків та виведення на консоль результатів роботи трьох функцій. Повторне виконання програми забезпечується за допомогою оператора goto та мітки m: .

Код цієї функції наведено на рисунку 3.4.

Зверніть увагу на різницю у використанні функцій f1() та f2().

Функція f1() повертає результат і тому звернення до неї можна використати безпосередньо для виведення результату на консоль:

cout << “y = ” << f1(a, b, x) << “\n”;

Функція f2() результат не повертає, тому її не можна використати безпосередньо для виведення результату на консоль. Спочатку функцію потрібно просто викликати, передавши в якості четвертого параметру змінну «y» до якої буде занесено результат:

f2(a, b, x, y);

Після цього отриманий результат можна вивести на консоль:

cout << “y = ” << y << “\n”;

```
//Тестування функцій f1, f2, f3
int main(){
   float a, b, x, y;
   m:  cout << "\n\nВведіть a, b, x \n";
      cin >> a >> b >> x;
      cout <<"Прийнято a="<<a<<"; b="<<b<< "; x="<<x<<"\n";
      cout << "\nФункція f1\n";
      cout << "y=" << f1(a,b,x)<<"\n";
      cout << "\nФункція f2\n";
      f2(a,b,x,y);
      cout << "y=" << y <<"\n";
      cout << "\nФункція f3\n";
      f3(a,b,x,y);
      cout << "y=" << y <<"\n";
    goto m;
    return (0);
   }
```

Рисунок 3.4 – Функція main() програмного файлу

3.2.5 Функція f1(), що повертає значення

У цій функції, що виконує розрахунки за формулою 3.1, попередньо обчислюється вираз *ax,* після чого обчислюється також числівник і знаменник. Для піднесення синусу до ступеня 3 та добування кореня ступеню b використовується макрос POW. Такий поділ обчислень на частини спрощує перевірку отриманих результатів. Код функції наведено на рисунку 3.5.

```
float f1(float a, float b, float x) {
    float z, ch, zn;
    z= a*x;
    ch = z+POW(sin(z),3);
    zn = POW(z/(1+x),1.0/b);
    return ch/zn;
}
```

Рисунок 3.5 – Функція, що повертає значення

3.2.6 Функція f2() типу void

Ця функція безпосередньо результат не повертає, але для збереження

результату використовується посилання &y на змінну з формальним ім'ям «y». Саме цій змінній і присвоюється результат обчислень. Код функції f2() наведено на рисунку 3.6.

```
void f2(float a, float b, float x, float &y) {
   float z, ch, zn;
   z= a*x;
   ch = z+POW(sin(z),3);
   zn = POW(z/(1+x),1.0/b);
   y=ch/zn;
}
```

Рисунок 3.6 – Функція, що повертає значення через параметр

3.2.7 Функція з виведенням проміжних результатів

Код функції f3() наведено на рисунку 3.7.

```
void f3(float a, float b, float x, float &y){
   float z, ch, zn;
   z= a*x;
   cout << "z=" << z <<"\n";
   ch = z+POW(sin(z),3);
   cout << "ch=" << ch <<"\n";
   zn = POW(z/(1+x),1.0/b);
   cout << "zn=" << zn <<"\n";
   y=ch/zn;
}
```

Рисунок 3.7 – Функція з виведенням проміжних результатів

3.3 ДОСЛІДЖЕННЯ СТВОРЕНОГО ПРОЕКТУ

3.3.1 Дослідження передачі параметрів за посиланням

Додайте до функції main() виведення значення змінної «y» до звернення до функції f3(). Запустіть програму на виконання та порівняйте значення «y» до і після звернення до функції f3().

Видаліть префікс & перед параметром «y» із списку параметрів функції f3() і знову проаналізуйте значення змінної до і після виклику f3(), а також значення проміжних результатів.

Відновіть заголовок процедури.

Зробіть висновки і зафіксуйте результати досліджень у звіті.

3.3.2 Анализ передачи параметров по значению

Змініть функцію f3() таким чином, щоб після обчислення результату значення формальних параметрів «a», «b» та «x» змінювалися, наприклад, ставали нулями.

Додайте до функції main() виведення значення відповідних фактичних параметрів після звернення до зміненої функції. Запустіть програму на виконання та з'ясуйте, чи зміняться значення фактичних параметрів після звернення до зміненої функції.

Відновіть текст функції.

Зробіть висновок і зафіксуйте в звіті.

3.4 ВИМОГИ ДО ЗВІТУ

- Назва роботи.
- Мета роботи.
- Короткі теоретичні відомості.
- Тексти функцій програми з коментарями.
- Результати тестування проекту у вигляді копій консолі.
- Результати дослідження способів передачі параметрів до функції
- Висновки.

3.5 КОНТРОЛЬНІ ПИТАННЯ

- Правила написання функцій.
- Поняття «функція» та правила визначення функції.
- Прототип функції. Завантаження прототипів бібліотечних функцій.
- Виклик функції.
- Способи передачі параметрів у функції.
- Області оголошення та доступу до імен.
- Макроси з параметрами.
- Дати пояснення до звіту.
- Написати функції за вимогою викладача.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программированя С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с
2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. –1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вільямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

4 ЛАБОРАТОРНА РОБОТА № 4.
ЛОГІЧНИЙ ТИП ДАНИХ І РОЗГАЛУЖЕННЯ У ПРОГРАМАХ

Мета роботи:
- Ознайомитися з операціями над даними логічного типу.
- Навчитися записувати та обчислювати логічні вирази.
- Ознайомитися з операторами **if … else** та **swith**.
- Створити проект, що реалізує алгоритми з розгалуженнями.

4.1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІІ

4.1.1 Логічний тип даних

У мові С такого типу не було, хоча поняття істинності та хибності, звичайно, були. Ці поняття відтворювалися за допомогою чисел. Істиною (true) вважалося довільне число, що не дорівнювало нулю, а хибність (false) позначалася нулем. Тільки у мові С++ з'явився тип bool, що використовується для даних, які можуть приймати тільки два значення – true і false. Але числові результати, як і раніше, у відповідному контексті теж можуть розглядатися як дані логічного типу, що інколи є причиною прикрих помилок.

Дані логічного типу за звичай з'являються як результат операцій порівняння. Наприклад, результатом обчислення виразу 2<3 буде true, а результат обчислення виразу sin(x)>0 може бути и true и false, в залежності від значення змінної х.

У таблиці 4.1 наведено операції порівняння, що визначені в мові С.

Таблиця 4.1 - Операції порівняння

Назва операції порівняння	Запис на мові С
Менше	<
Менше або дорівнює	<=
Більше	>
Більше або дорівнює	>=
Дорівнює	==
Не дорівнює	!=

4.1.1.1 Операції над даними логічного типу

Для даних типу bool визначені дві бінарних операції – «і» (&&) та «або» (||). Результати застосування цих операцій до можливих комбінацій логічних операндів наведені в таблиці 4.2. Цю таблицю слід знати, так само як і таблицю множення.

Крім бінарних, визначена одна унарна операція - «ні». У мові С ця операція позначається знаком оклику (!). Операція змінює значення логічної змінної на протилежне. Наприклад, результатом обчислення виразу !(2> 3) буде **true**, а результат обчислення виразу !(sin (3.1416 / 2) <> 0) буде **false**.

Таблиця 4.2 – Результати виконання логічних операцій

Перший операнд	Другий операнд	Логічі операції	
		&& (і)	|| (або)
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

4.1.1.2 Логічні вирази

Вирази, в яких використовуються операнди логічного типу та операції над ними, називаються логічними. Результатом обчислення такого виразу може бути тільки true або false. Частинами логічного виразу можуть бути арифметичні вирази, які беруть участь в операціях порівняння.

Записуючи логічні вирази слід враховувати старшинство операцій. Пріоритети операцій, які можуть брати участь в логічних виразах, наведені в таблиці 4.3.

.

Таблиця 4.3 – Пріоритети операцій

Операції	Пріоритет
* / %	3
+ -	4
<< >>	5
< > <= >=	6
= = !=	7
& ^ | && ||	8,9,10,11,12

Якщо доводиться в першу чергу обчислювати операції які мають найнижчий пріоритет, то такі операції беруться в дужки.

Слід також брати до уваги, що логічні операції в С не обов’язково обчислюються повністю. Як тільки результат стає однозначним, подальші обчислення припиняються.

4.1.2 Алгоритми з розгалуженнями

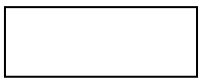
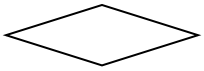
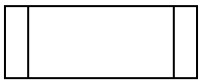
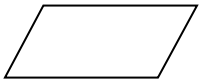
У повсякденному житті ми часто зустрічаємося з такими видами алгоритмів. Наприклад, якщо йде дощ, ми беремо парасольку, а якщо мороз, надягаємо теплу куртку. Таким чином, у алгоритмі, що має розгалуження, деякі дії можуть виконуватися тільки за певних умов.

При розробці алгоритмів, що розгалужуються, корисно зображувати схеми алгоритмів, використовуючи спеціальні позначення. Схема алгоритму являє собою як би план написання програми. Складання таких схем не вимагає багато часу, але істотно підвищує продуктивність праці програміста.

Деякі умовні позначення, що використовуються при складанні схем

алгоритмів, наведені в таблиці 4.4. При зображенні алгоритму окремі блоки нумеруються і з'єднуються лініями. Стрілки використовуються тільки для зворотних напрямків (вгору і ліворуч).

Таблиця 4.4 - Умовні позначення для схем алгоритмів

	Начало и кінець алгоритму
	Обробка інформації, наприклад, розрахунок за формулою
	Перевірка умови і прийняття рішення. Після цього блоку можливі різні шляхи продовження виконання алгоритму
	Зумовлений процес, наприклад, звернення до функції.
	Виведення або введення інформації.

В якості прикладу розглянемо схеми алгоритму розв’язання квадратного рівняння $ax^2 + bx + c = 0$.

На рисунку 4.1 зображено схему алгоритму, де аналізується перший коефіцієнт рівняння і приймається рішення, чи є рівняння квадратним, або воно лінійне. У першому випадку буде викликана процедура вирішення квадратного рівняння, у другому - процедура рішення лінійного рівняння.

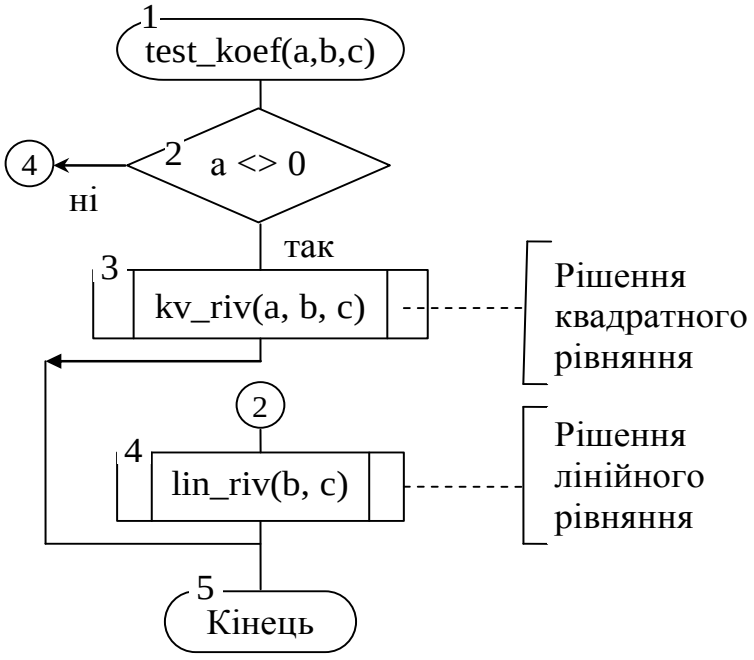


Рисунок 4.1- Схема алгоритму аналізу коефіцієнтів квадратного рівняння

На рисунку 4.2 зображена схема алгоритму рішення лінійного рівняння. У цьому алгоритмі аналізуються значення решти коефіцієнтів. Якщо обидва вони дорівнюють нулю, то рівняння 0x + 0 = 0 задовольняє будь-яке значення x. Якщо ж b дорівнює 0, а c не дорівнює 0, то рівняння рішення не має. В інших випадках корінь рівняння визначається за формулою r =-c / b.

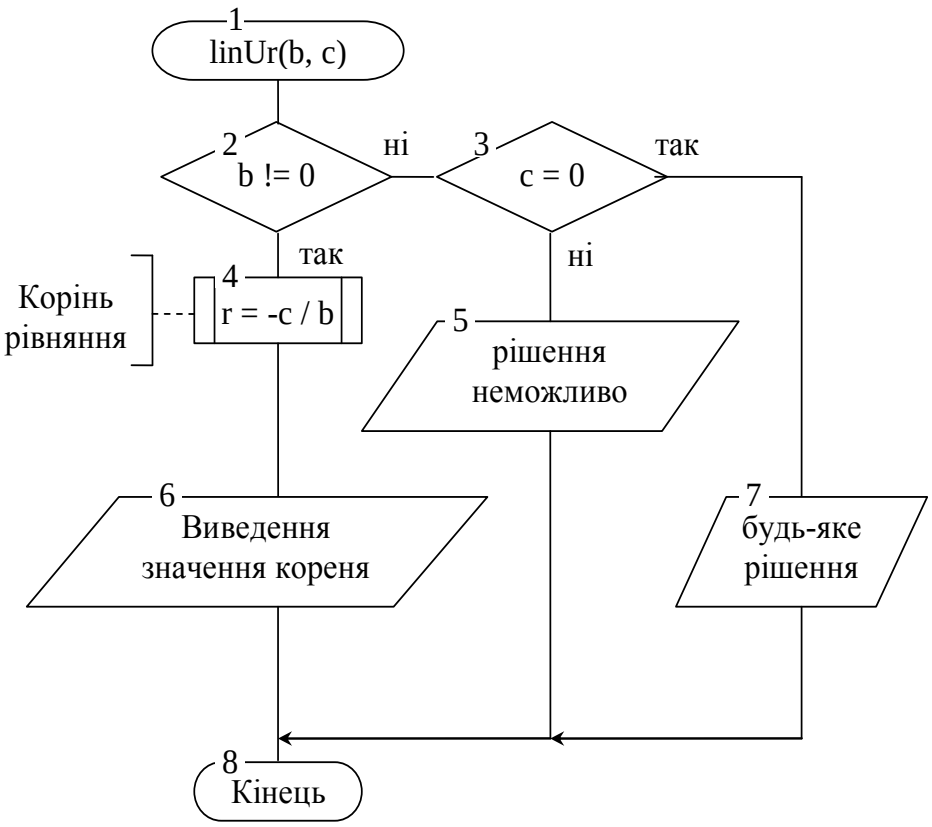


Рисунок 4.2 – Схема алгоритму розв'язання лінійного рівняння

Схему алгоритму розв'язання квадратного рівняння, що повинен виконуватися, якщо коефіцієнт «a» не дорівнює 0, слід скласти самостійно і привести в звіті. Алгоритм повинен передбачати аналіз дискримінанта та виведення значень дійсних або комплексних коренів.

4.1.3 Програмування розгалужень

4.1.3.1 Оператор розгалуження if…else

Оператор if ... else дозволяє вибрати один з двох можливих варіантів виконання програми. Синтаксис запису цього оператора представлений на рисунку 4.3.

```
if ( <умова> )
        <оператор 1>
else    <оператор 2>
```

Рисунок 4.3– Синтаксис оператора **if … else**

Виконується цей оператор наступним чином:

– обчислюється значення умови (умова - вираз, результат якого приводиться до логічного типу);

– якщо результат приймає значення “true”, то виконується <оператор 1>;

– якщо результат приймає значення “false”, то виконується <оператор 2>, що розташований за словом else.

Як приклад використання цього оператора можна привести програмну реалізацію схеми алгоритму зображеного на рисунку 4.4.

```
void test_koef(float a, float b, float c) {
  if (a==0)
     // Звернення до функції рішення лінійного
     lin_riv(b,c);
  else
     // Звернення до функції рішення квадратного рівняння
     kv_riv(a,b,c);
}
```

Рисунок 4.4 – Програмна реалізація алгоритму аналізу коефіцієнтів квадратного рівняння

Якщо замість одного оператора потрібно виконати декілька, їх слід об’єднати у блок за допомогою фігурних дужок. У цьому випадку крапка з комою після закриваючої дужки не ставиться.

У деяких випадках, при невиконанні умови, що перевіряється, робити нічого не треба, тобто <оператор 2> не потрібен. У цьому випадку оператор if можна застосовувати в скороченій формі. Синтаксис скороченої форми має вигляд, представлений на рисунку 4.5.

```
if (<умова>)
      <оператор>
```

Рисунок 4.5– Скорочена форма оператора **if**

Оператор **if**, як повний так і скорочений, може бути вкладений у інший оператор **if**. При цьому слід пам’ятати, що кожна else-частина пов’язується з найближчим **if**. Якщо ж else-частина відноситься до одного з попередніх **if**, то слід застосовувати фігурні дужки. Приклад використання вкладених операторів **if** наведено на рисунку 4.6.

Стандарт мови С гарантує можливість 15 рівнів вкладення для оператора **if**. Проте не слід цим зловживати. Вкладені оператори **if** виглядають зазвичай досить заплутано і часто є причиною виникнення логічних помилок.

```
if  <умова1>
{
        < оператор 1.1>;
        if  <умова2>
        {
                <оператор 2.1>;
        }
        else // для умови 2
        {
                <оператор 2.2>;
        }
        < оператор 1.2>;
}
else // для умови 1
}
        < оператор 1.3>;
}
```

Рисунок 4.6– Приклад використання вкладених операторів **if**

Більш зручними і наочними є ланцюжки повних операторів **if**, в яких після кожного слова **else** (за винятком останнього) знову йде повний оператор **if,** рисунок 4.7. Такі ланцюжки зручно використовувати при вирішенні багатоваріантних задач. Вони досить легко аналізуються, за структурою подібні до розглянутого нижче оператору **swith**, і їх можна застосовувати, на відміну від нього, для перевірки будь-яких умов.

```
if  <умова1>
        <оператор 1>
else if  <умова2>
        <оператор 2>
else if  <умова3>
        <оператор 3>
…
else <оператор N>
```

Рисунок 4.7– Приклад ланцюжка повних операторів **if**

4.1.3.2 Умовна операція

Цю операцію називають ще теренарною операцією. Вона виконується над трьома операндами і записується так, як показано на рисунку 4.8.

<умова> ? < вираз 1 > : < вираз 2 >

Рисунок 4.8 – Синтаксис теренарного оператора

Тут 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. На рисунку 4.9 наведено приклад використання вкладеного теренарного оператора у функції для розрахунку стипендії.

```
float stip(bool budget, float ball){
    return !budget || ball<4 ? 0 : ball<5 ? 600 : 800;
}
```

.

Рисунок 4.9– Приклад використання теренарного оператора

4.1.3.3 Оператор вибору swith

Конструкція **swith** дозволяє ефективно реалізувати численні розгалуження у тих випадках, коли вибір визначається значеннями змінної порядкового типу (int, char).

Синтаксис оператору **swith** представлено на рисунку 4.10.

```
swith ( <вираз> ) {
        case <константа 1>:
        <оператор 1>
        case < константа 2>:
        <оператор 2>;
…
        case< константа n>:
        <оператор n>;
default:
        <оператор n+1>;
}
```

Рисунок 4.10 – Синтаксис оператора **swith**

У цьому описі < вираз > - це вираз, значення якого визначає подальше виконання оператора. В окремому випадку < вираз > може бути просто змінною. Результат обчислення виразу може бути тільки ціле число або символ.

<константа > – це мітка даного варіанту, яка може бути константою або виразом. Значенням константи або результатом обчислення виразу також має

бути ціле число або символ. Порядок запису міток довільний, але вони мають бути різними. Останній варіант, що починається словом default, не є обов'язковим. Мітка відділяється від оператора двокрапкою.

< оператор > визначає дії, які повинні бути виконані, якщо < вираз > приймає значення константи. В якості оператора може використовуватися і складений оператор.

Порядок виконання оператора **swith** описаний нижче:

– спочатку обчислюється значення виразу;

– далі, 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.

В якості прикладу (рисунок 4.11) розглянемо функцію, що опрацьовує номер варіанту вибраного з меню і відповідно до цього задає коефіцієнти квадратного рівняння. Ця функція може стати у нагоді під час тестування програми обчислення коренів квадратного рівняння. Внаслідок того, що кожний із варіантів виконується окремо і не пов'язаний з іншими, виникає необхідність кожен варіант завершувати оператором break.

Варіант default у цьому прикладі не використовується.

```
void run_variant(int v){
    switch(v) {
    case 0:// Завершити програму
           exit(0); break;
    case 1:// Задати коефіцієнти користувача
           user_koef(); break;
    case 2:// Дійсні корені
           test_koef(4, 5, -3); break;
    case 3:// Комплексні корені
           test_koef(4, 2, 3); break;
    case 4:// Лінійне рівняння
           test_koef(0, 2, 3); break;
    case 5:// Будь-яке рішення
           test_koef(0, 0, 0); break;
    case 6:// Нема розв'язку
           test_koef(0, 0, 5);
    }
}
```

Рисунок 4.11 – Приклад використання оператора **swith**

4.1.4 Оператор переходу goto

Цей оператор забезпечує перехід до іншого оператора, що позначений

заданою міткою. Мітка, до якої відбувається перехід може бути розташована як до так і після оператора **goto**. Синтаксис оператора **goto** наведено на рисунку 4.12.

```
goto  <мітка> ;
        …
<мітка>:<оператор>
```

Рисунок 4.12– Синтаксис оператора **goto**

Оператор **goto** використовується рідко, тому що він заплутує програму і робить її малозрозумілою.

4.2 РЕАЛИЗАЦІЯ ПРОЕКТУ «IF_SWITCH»

У цьому проекті студент повинен реалізувати алгоритм розв'язання квадратного рівняння. Необхідно передбачити обробку будь-яких комбінацій значень коефіцієнтів рівняння. Проект має також передбачати тестування усіх можливих варіантів роботи алгоритму.

4.2.1 Початковий інтерфейс проекту

Після запуску програми на консолі має з’явитися перелік варіантів роботи з програмою і пропозиція користувачеві (prompt) вибрати один із варіантів. На рисунку 4.13 наведено вигляд консолі після запуску програми.

```
Тестування програми розв’язку квадратного рівняння
Виберіть один із варіантів
0. Завершити програму.
1. Задати коефіцієнти користувача.
2. Дійсні корені.
3. Комплексні корені.
4. Лінійне рівняння.
5. Будь-яке рішення.
6. Нема розв'язку.
Ваш вибір:
```

Рисунок 4.13 – Вигляд консолі після запуску програми

Наведений перелік варіантів надає можливість протестувати реалізацію різних гілок алгоритму розв’язання квадратного рівняння і порівняти отримані результати з розрахунковими.

4.2.2 Допоміжні файли

На рисунку 4.14 наведено перелік файлів, які необхідно включити до програми, щоб забезпечити її нормальне функціонування.

```
#include <iostream>
#include <cmath>
#include <cstdlib>

using namespace std;
```

Рисунок 4.14 – Директиви #include програмного файлу

Файл iostream забезпечує реалізацію обміну інформацією з консоллю.
Файл cmath забезпечує доступ до математичних функцій.
Файл cstdlib забезпечує доступ до функції exit().

4.2.3 Прототипи функцій

На рисунку 4.15 наведено перелік функцій, які будуть використовуватися у файлі програми.

```
void run_variant(int v);
void user_koef();
void test_koef(float a, float b, float c);
void kv_riv(float a, float b, float c);
void lin_riv(float b, float c);
```

Рисунок 4.15 – Прототипи функцій програмного файлу

Функція run_variant(int) опрацьовує номер варіанту вибраного з меню і відповідно до цього задає коефіцієнти квадратного рівняння. Текст цієї функції наводився вище, як приклад використання оператору switch.

Функція user_koef() забезпечує введення з консолі довільних значень коефіцієнтів квадратного рівняння.

Функція test_koef(float, float, float) забезпечує аналіз коефіцієнтів квадратного рівняння та прийняття рішення про виклик функцій розв’язання або квадратного рівняння, або лінійного.

Функція kv_riv(float, float, float) забезпечує розв’язання квадратного рівняння.

Функція lin_riv(float, float) забезпечує розв’язання лінійного рівняння.

4.2.4 Функція main() програмного файлу

Функція main() програмного файлу забезпечує виведення на консоль меню варіантів і введення вибору користувача. Для обробки вибору користувача викликається функція run_variant(int). Повторне виведення меню після завершення обробки забезпечується за допомогою оператора goto і мітки begin.

Код цієї функції наведено на рисунку 4.16.

```
int main(){
   begin:
    cout << "\n\n Тестування програми розв'язку"
         << "  квадратного рівняння \n"
         << "  Виберіть один із варіантів \n"
         << "  0. Завершити програму. \n"
         << "  1. Задати коефіцієнти користувача.\n"
         << "  2. Дійсні корені. \n"
         << "  3. Комплексні корені. \n"
         << "  4. Лінійне рівняння. \n"
         << "  5. Будь-яке рішення. \n"
         << "  6. Нема розв'язку. \n"
         << "  Ваш вибір:";
    int v;
    cin >> v;
    run_variant(v);
    goto begin;
    return 0;
}
```

Рисунок 4.16 – Функція main() програмного файлу

4.2.5 Функція обробки номеру варіанта

Ця функція за допомогою оператора swith обробляє номер варіанта, що вибрав користувач. Код функції run_variant(int) наведено на рисунку 4.11.

4.2.6 Функція введення коефіцієнтів рівняння за вибором користувача

Функція user_koef() забезпечує введення з консолі довільних значень коефіцієнтів квадратного рівняння. Після цього викликається функція аналізу коефіцієнтів квадратного рівняння. Код функції наведено на рисунку 4.17.

```
void user_koef(){
    double a, b, c;
    cout << " Введіть коефіцієнти рівняння a, b, c \n";
    cin >> a >> b >> c;
    test_koef(a, b, c);
}
```

Рисунок 4.17 –Функція user_koef() програмного файлу

4.2.7 Функція для розв'язання лінійного рівняння

Функція lin_riv(float, float) забезпечує розв'язання лінійного рівняння. Алгоритм роботи цієї функції наведено на рисунку 4.2. Завдання студента – написати відповідну функцію і включити її до складу проекту.

4.2.8 Функція для розв'язання квадратного рівняння

Функція kv_riv(float, float, float) має забезпечити розв'язання квадратного рівняння. Завдання студента – розробити схему алгоритму, написати відповідну функцію і включити її до складу проекту.

Схема алгоритму рішення квадратного рівняння повинна мати дві вітки, одна з яких забезпечує обчислення і виведення значень дійсних корнів, а друга – комплексних корнів. Вибір слід робити на підставі значення дискримінанту.

4.3 ВИМОГИ ДО ЗВІТУ

- Назва роботи.
- Мета роботи.
- Короткий опис логічного типу даних і правил запису логічних виразів.
- Коротка характеристика операторів **if, switch**.
- Схеми алгоритмів вирішення квадратного рівняння.
- Тексти функцій програми з коментарями.
- Результати тестування проекту у вигляді копій консолі.
- Висновки.

4.4 КОНТРОЛЬНІ ПИТАННЯ

- Запис логічних виразів і правила їх обчислення.
- Основні логічні операції і таблиці для обчислення результатів цих операцій.
- Правила складання схем алгоритмів, що розгалужуються.
- Інструкція **if** та її варіанти. Приклади.
- Інструкція switch та приклад її використання.
- Пояснення текстів функцій програми.
- Написати підпрограму з розгалуженнями за завданням викладача, наприклад, функцію для визначення максимального (мінімального, середнього) з двох (трьох) чисел.
- Написати підпрограму обчислення стипендії в залежності від статусу студента (бюджет чи ні) та середнього балу.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программированя С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с
2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. –1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вільямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

5 ЛАБОРАТОРНА РОБОТА № 5. ПОБУДОВА ЦИКЛІВ З ОПЕРАТОРМИ „WHILE” І „DO...WHILE”

Мета роботи:

– Познайомитися з особливостями та принципами використання циклів **while** та **do…while**.
– Створити проект, що забезпечує вирішення математичних задач чисельними методами.

5.1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

5.1.1 Циклічні алгоритми

Алгоритми вирішення багатьох задач є циклічними, тобто для досягнення результату певна послідовність дій повинна бути виконана декілька разів.

Наприклад, для того щоб знайти прізвище людини у списку, треба перевірити перше прізвище списку, потім друге, третє і т. д. доти, поки не буде знайдено потрібне прізвище або не закінчиться список.

Алгоритм, в якому є послідовність операцій (група операторів), яка повинна бути виконана декілька разів, називається циклічним, а сама послідовність операцій іменується тілом циклу.

У програмах на мові С цикл може бути реалізований за допомогою операторів while, do...while і for. Цикл, який створюється за допомогою оператора for, буде розглянуто в наступній роботі. Поки ж ми розглянемо оператора while і do... while.

Особливість циклів, що створюються за допомогою цих операторів, в тому, що в них заздалегідь не відомо, скільки разів буде виконуватися тіло циклу. Виконання повторюється, поки задовольняється деяка умова. А параметри, що впливають на умову змінюються у тілі циклу.

Типовими прикладами використання таких циклів є обчислення із заданою точністю, пошук у масиві або у файлі.

5.1.2 Оператор while

Особливість ццого оператора полягає у тому, що умова перевіряється перед виконанням тіла циклу, тому цикл while називають циклом з передумовою.

В узагальненому вигляді оператор while записується так (рис.5.1).

```
while( умова виконання )
{
        оператори тіла циклу;
}
```

Рисунок 5.1 – Синтаксис оператора **while**

На цьому рисунку <умова виконання > - це вираз логічного типу, що визначає умову за якої виконуються <оператора тіла циклу >.

Загалом, оператор **while** виконується у такий спосіб:

– обчислюється значення виразу <умова виконання>;

– якщо значення виразу <умова виконання > дорівнює **false** або 0, тобто умова не виконується, виконання <операторів тіла циклу > припиняється;.

– якщо значення виразу <умова виконання дорівнює **true** або не 0 (умова виконується), то виконуються <оператора тіла циклу >, розташовані між фігурними дужками;

– після цього знову все повторюється.

Слід зауважити, що для того щоб цикл завершився, потрібно щоб послідовність операторів, розташованих між фігурними дужками, впливала на значення <умови виконання >.

На рисунку 5.2 представлено схему алгоритму виконання цього циклу.

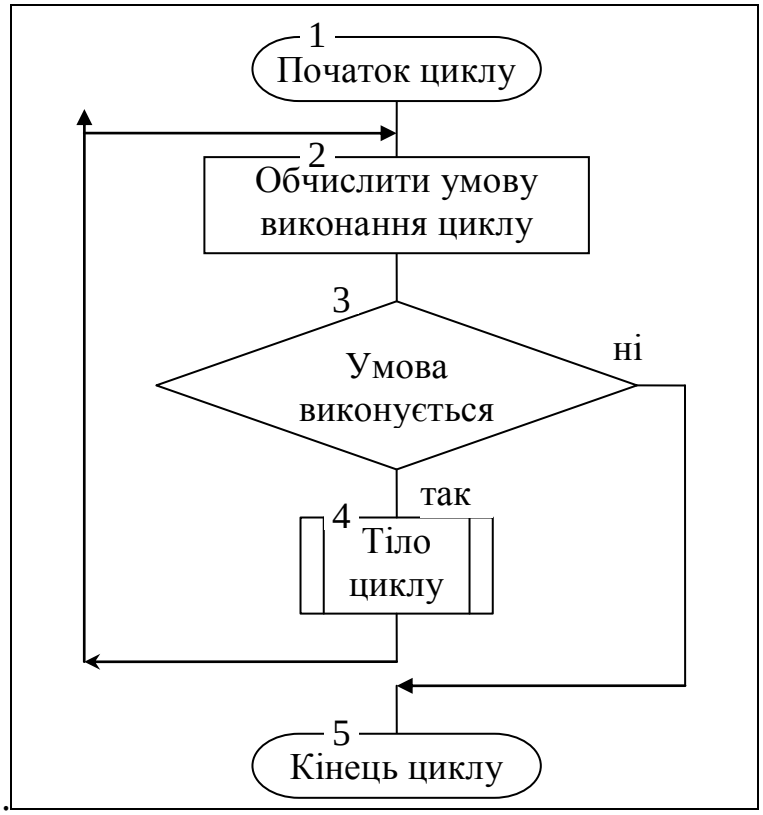


Рисунок 5.2 – Схема алгоритму виконання циклу while

Як приклад розглянемо функцію, що підраховує суму цифр цілого числа. Виділяти окремі цифри числа, починаючи з останньої можна за допомогою операції %10. Далі число можна поділити на 10, що призведе до втрати останньої цифри і передостання цифра стане останньою. Після цього операцію %10 можна повторити, і так далі, доки число після чергового ділення на 10 не стане нулем.

Текст цієї функції наведено нижче:

```
uint sumFig(int x){
    int sum=0;
    while(fabs(x)>0){
        sum+=x%10;
        x/=10;
    }
    return sum;
}
```

У мові С цикл while може виглядати дещо незвично, якщо у виразі для перевірки умови використовувати операцію присвоєння, або операцію «,» (кома). Операція кома пов'язує декілька виразів, що розглядаються як один. Результатом такої послідовності буде результат обчислення останнього виразу.

Використання таких можливостей може призвести до скорочення тіла циклу і навіть до його зникнення, так як це відбулося у наступній функції, яка робить те саме, що й попередня.

```
uint sumFig(int x){
    int sum=0;
    while(sum+=x%10, fabs(x/=10)>0);
    return sum;
}
```

5.1.3 Оператор do…while

Особливість цього оператора полягає у тому, що умова перевіряється після виконання тіла циклу, внаслідок чого <оператори тіла циклу > виконуються, принаймні, один раз. Тому цикл **do…while** називають циклом з постумовою.

У мові програмування С оператор **do…while** виглядає таким чином (рисунок 5.3):

do {
операторы тіла циклу
} while (*умова повтору тіла циклу*)**;**

Рисунок 5.3 – Синтаксис оператора **do…while**

На рисунку <умова повтору тіла циклу > - це вираз логічного типу, що визначає умову за якої будуть знов виконані <оператора тіла циклу >.

Цикл виконується таким чином:

– спочатку виконуються <оператори тіла циклу >, розташовані між фігурними дужками;.

– потім обчислюється значення виразу <умова повтору тіла циклу >.

– якщо значення виразу <умова повтору тіла циклу > дорівнює **true** або не 0 (умова виконується), то знову виконуються <оператори тіла циклу >, розташовані між фігурними дужками;

– якщо значення виразу <умова виконання > дорівнює **false** або 0, тобто умова не виконується, виконання <операторів тіла циклу > припиняється.

Схема алгоритму виконання цього циклу представлена на рисунку 5.4.

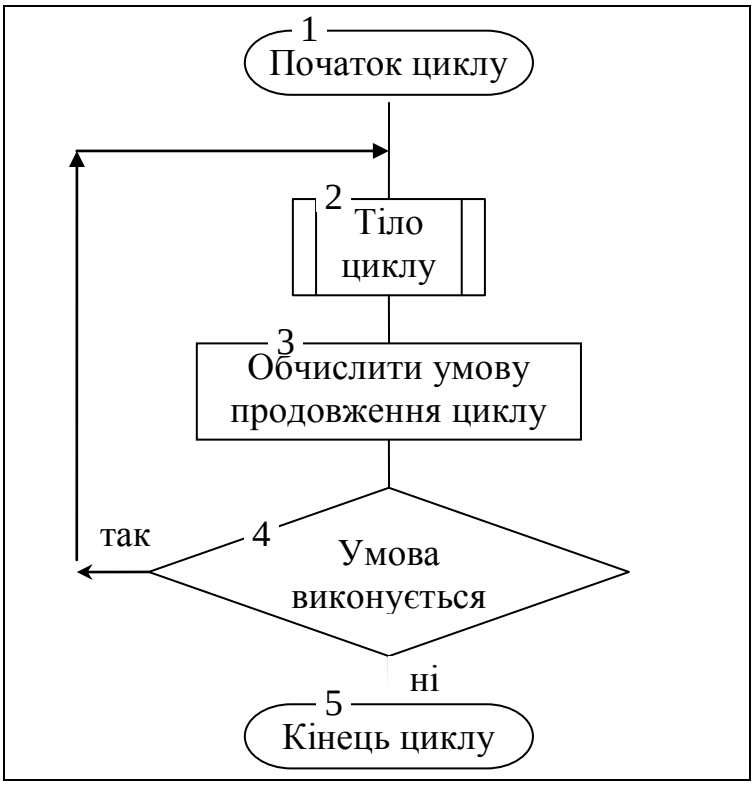


Рисунок 5.4 – Схема алгоритму виконання циклу **do…while**

Як приклад використання циклу **do…while** розглянемо програму, вхід до якої контролюється паролем. Цикл **do…while** тут буде доречним, бо спочатку потрібно ввести пароль, а потім перевірити його правильність. Якщо пароль не правильний, то введення паролю слід повторити.

Текст цієї програми наведено нижче.

```
string p="qwerty";
string inpt;
do{
   cout<<"/nВведіть пароль ";
   cin>>inpt;
}while(p!=inpt);
cout<<"OK!\n";
 …
```

5.1.4 Переривання циклу

Під час програмування деяких задач виникає потреба перервати виконання циклу, не чекаючи виконання умов виходу з циклу. Така можливість забезпечується оператором break. Цей оператор може бути розташований у будь-якому місці тіла циклу. В результаті цикл відразу ж припиняється.

На відміну від break оператор continue перериває виконання тільки тіла

циклу і одразу ж переходить до обчислення умови продовження циклу.

Крім цих операторів можна використовувати оператор return, який перериває виконання не тільки циклу, але і всієї функції.

Можна використовувати і оператор exit, але в цьому випадку завершиться робота головної програми.

5.1.5 Ітераційні алгоритми

Алгоритми називають ітераційними, якщо в них багаторазово повторюються обчислення за однією і тією ж формулою, причому отриманий результат використовується в якості вихідних даних для наступного розрахунку. Обчислення повторюються доти, поки не буде виконана деяка умова.

У таблиці 5.1 наведені приклади рівнянь, які можуть бути вирішені методом ітерацій, і відповідні ітераційні формули. Спосіб отримання ітераційної формули дуже простий. Рівняння вирішується відносно невідомої змінної, причому в праву частину ітераційної формули входить та ж невідома змінна.

Таблиця 5.1 Рівняння, які вирішуються методом ітерацій

Варіант	Рівняння	Ітераційна формула	Обмеження
1	2	3	4
0	$a \cdot e^{-b \cdot x} - x = 0$	$x = a \cdot e^{-b \cdot x}$	0<ab<1, x0>0
1	$a \cdot x^2 - b \cdot \sin(x) = 0$	$x = \sqrt{\frac{b \cdot \sin(x)}{a}}$	a>0, b>0, b/a<6, x0<π/2
2	$\sqrt{a \cdot x} + \sqrt{b \cdot x} - cx = 0$	$x = \frac{\sqrt{a \cdot x} + \sqrt{b \cdot x}}{c}$	a>0, b>0, c>0, x0>0
3	$arctg(a \cdot x) - bx = 0$	$x = \frac{arctg(a \cdot x)}{b}$	a>1, b>0, x0>0
4	$a \cdot \sin(b \cdot x) - c \cdot x = 0$	$x = \frac{a \cdot \sin(b \cdot x)}{c}$	ab>c, |x0|<π/(2b)
5	$e^{-a \cdot x} + b \cdot x - c = 0$	$x = \frac{c - e^{-a \cdot x}}{b}$	a>0, b>0 c>1, x0>0
6	$\frac{a}{x} - x + c = 0$	$x = \frac{a}{x} + c$	a>0, c>0, x0>0
7	$\ln(a \cdot x) - bx + c = 0$	$x = \frac{\ln(a \cdot x) + c}{b}$	a>1, c>1, b<c, x0>1
8	$\sqrt{a \cdot x} - b \cdot x + c = 0$	$x = \frac{\sqrt{a \cdot x} + c}{b}$	a>0, b>0, c>0, x0>0
9	$a \cdot e^{-x} - b \cdot x^2 + c = 0$	$x = \frac{\sqrt{c + a \cdot e^{-x}}}{b}$	a>0, b>0, c>0, x0>0

Використання ітераційних алгоритмів дозволяє розв’язувати, наприклад, трансцендентні рівняння. Однак ці методи не є універсальними. Їх можна застосовувати лише тоді, коли результати послідовних ітерацій сходяться, тобто поступово наближаються до деякого значення, яке і буде рішенням рівняння. Алгоритм розв'язання рівнянь, наведених у таблиці 5.1 полягає в наступному.

Береться якесь наближене до розв’язку значення x0 (початкове наближення) і підставляється до ітераційної формули. Отримане за ітераційною формулою нове приблизне значення розв’язку порівнюється з попереднім. Якщо ці значення істотно відрізняються один від одного, то нове наближене значення підставляється в ітераційну формулу замість старого і на його основі отримують нове наближене значення. Так триває доти, поки нове і старе наближення стануть достатньо близькими один до одного.

5.1.5.1 Приклад ітераційного алгоритму для обчислення кубічного кореня

Прикладом ітераційного алгоритму може служити алгоритм обчислення кубічного кореня методом Ньютона, який полягає у послідовному обчисленні наближених значень кореня за формулою 5.1.

$$x_{new} = x_{old} - \frac{x_{old}^{3} - a}{3 \cdot x_{old}^{2}} \qquad (5.1)$$

де a – число, з якого добувається корінь, x_{old} – попереднє наближене значення кореня, x_{new} – наступне, більш точне, ніж x_{old} значення кореня.

Таким чином, формула 5.1 дозволяє послідовно уточнювати значення кореня, використовуючи попередній результат. В якості початкового значення для x можна взяти число, з якого добувається корінь.

Зазвичай обчислення проводяться доти, поки різниця між двома послідовними наближеннями за модулем не стане менше деякого, наперед заданого, достатньо малого числа.

В алгоритмі передбачено переривання циклу за допомогою оператора break в тому випадку, якщо змінна Xold дорівнює нулю, для того, щоб виключити ділення на 0.

Схему алгоритму, що реалізує цей метод, наведено на рисунку 5.5. Для реалізації цього алгоритму використовується цикл **do…while**.

5.1.6 Алгоритми обчислення сум нескінченних рядів

У цих алгоритмах послідовно підсумовуються члени нескінченного ряду. Накопичення суми має сенс тільки у тому випадку, якщо ряд сходиться, тобто значення членів ряду поступово зменшуються. Накопичення суми проводять доти, поки черговий член ряду не стане менше деякого, наперед заданого, достатньо малого числа. Подібність таких алгоритмів з попереднім полягає в тому, що обчислення кожного наступного члена ряду проводиться за значенням попереднього.

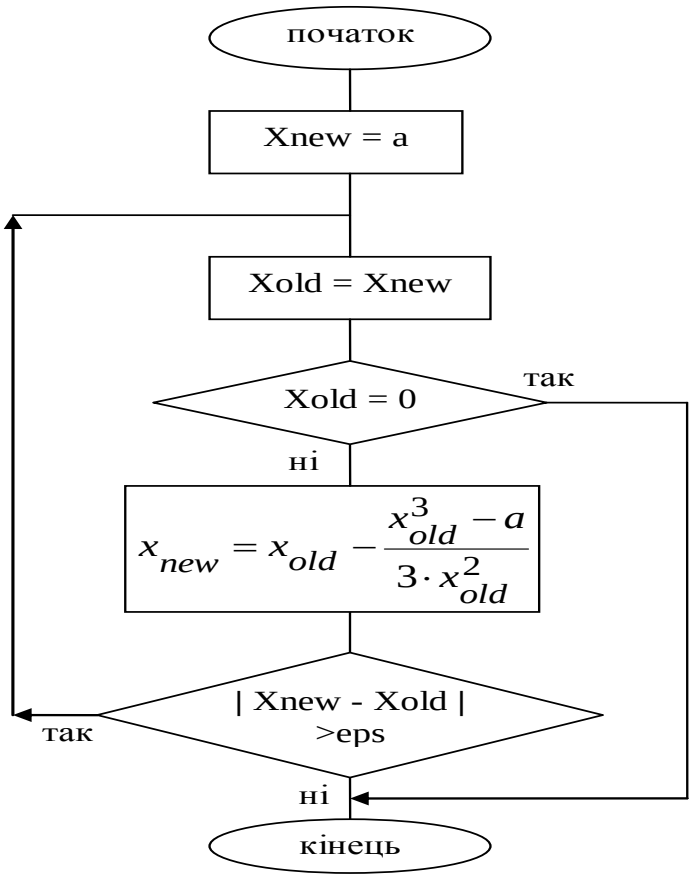


Рисунок 5.5 – Схема алгоритму знаходження кубічного кореня

В якості прикладу обчислення суми ряду розглянемо алгоритм обчислення синуса деякого числа. Синус можна представити як суму нескінченного ряду 5.2.

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^{2 \cdot i - 1}}{(2 \cdot i - 1)!} = \sum_{i=1}^{\infty} U_i \qquad (5.2)$$

Особливість даного ряду полягає в тому, що кожен член ряду, починаючи з другого, може бути знайдений з попереднього за формулою 5.3.

$$U_i = -U_{i-1} \cdot \frac{x^2}{(2 \cdot i - 2) \cdot (2 \cdot i - 1)} \qquad (5.3)$$

Алгоритм обчислення синуса за формулою 5.2 з використанням співвідношення 5.3 представлений на рисунку 5.6.

Слід мати на увазі, що ряд 5.2 при великих значеннях x сходиться повільно, а факторіал і «x в ступені» ростуть дуже швидко. Це призводить до того, що значущі цифри цих чисел перестають міститися в розрядній сітці і, отже, обрізаються, внаслідок чого результат спотворюється.

Для усунення цього недоліку в розглянутий алгоритм перед основним циклом накопичення суми ряду варто було б додати додатковий цикл. У цьому циклі змінна x повинна зменшуватися кожен раз на величину періоду синуса,

який дорівнює 2π. Цикл працює доти, поки x не стане менше, ніж 2π.

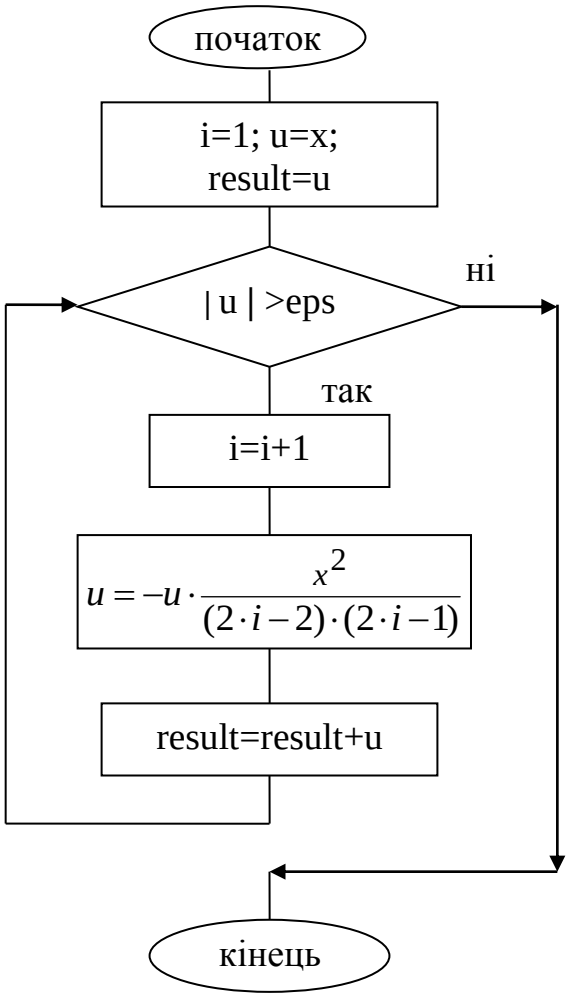


Рисунок 5.6 – Алгоритм обчислення синуса

Таблиця 5.2 Завдання на обчислення сум нескінченних рядів

№	Функція	Ряд	Рекурентна формула
1	2	3	4
0	e	$2+\frac{1}{2!}+\frac{1}{3!}+...=2+\sum_{i=2}^{\infty} u_i$	$u_{i+1}=\frac{u_i}{i+1}$
1	e^x	$1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+...=\sum_{i=0}^{\infty} u_i$	$u_i = u_{i-1} \cdot \frac{x}{i}$, для x<1
2	$\frac{1}{1-x}$	$1+x+x^2+.x^3..=\sum_{i=0}^{\infty} u_i$	$u_i = u_{i-1} \cdot x$, для x<1
3	$\frac{x}{(1-x)^2}$	$x+2 \cdot x^2+.3 \cdot x^3..=\sum_{i=1}^{\infty} i u_i$	$u_i = u_{i-1} \cdot x$, для x<1

Продовження таблиці 5.2

1	2	3	4
4	$\frac{x}{x-1}$	$1+\frac{1}{x}+\frac{1}{x^2}+\frac{1}{x^3}...=\sum_{i=0}^{\infty}u_i$	$u_i=\frac{u_{i-1}}{x}$, для x>1
5	cos(x)	$1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+...=\sum_{i=0}^{\infty}u_i$	$u_i=-u_{i-1}\cdot\frac{x^2}{(2\cdot i-1)\cdot 2\cdot i}$, для x<π
6	sh(x)	$x+\frac{x^3}{3!}+\frac{x^5}{5!}+...=\sum_{i=1}^{\infty}u_i$	$u_i=u_{i-1}\cdot\frac{x^2}{(2\cdot i-2)\cdot(2\cdot i-1)}$ для x<1
7	arctg(x)	$x-\frac{x^3}{3}+\frac{x^5}{5}-\frac{x^7}{7}+...=\sum_{i=0}^{\infty}u_i$	$u_i=-u_{i-1}\cdot\frac{x^2\cdot(2\cdot i-1)}{2\cdot i+1}$ для x<1
8	ln(1+x)	$x-\frac{x^2}{2}+\frac{x^3}{3}+...=\sum_{i=1}^{\infty}u_i$	$u_i=-u_{i-1}\cdot\frac{x\cdot(i-1)}{i}$, для x<1
9	$\frac{\sin(x)}{x}$	$1-\frac{x^2}{3!}+\frac{x^4}{5!}-\frac{x^6}{7!}+...=\sum_{i=0}^{\infty}u_i$	$u_i=-u_{i-1}\cdot\frac{x^2}{2\cdot i\cdot(2\cdot i+1)\cdot}$, для x<π

5.2 СТВОРЕННЯ ПРОЕКТУ «WHILE_DO»

У цьому проекті студент повинен реалізувати алгоритми розв'язання рівняння методом ітерацій та обчислення суми нескінченного ряду, що вимагають використання операторів **while** або **do ... while**.

В якості зразка ми будемо розглядати створення проекту для обчислення кубічного кореня та обчислення значення функції синус. Алгоритми цих обчислень розглядалися вище.

Завдання студента - реалізувати подібним чином алгоритми, відповідні його варіанту.

5.2.1 Початковий інтерфейс проекту

Після запуску програми на консолі має з’явитися перелік варіантів роботи з програмою і пропозиція користувачеві (промпт) вибрати один із варіантів. На рисунку 5.7 наведено вигляд консолі після запуску програми.

Наведений перелік варіантів надає можливість протестувати реалізацію розглянутих вище алгоритмів і порівняти отримані за їх допомогою результати із результатами, що отримані за допомогою стандартних засобів.

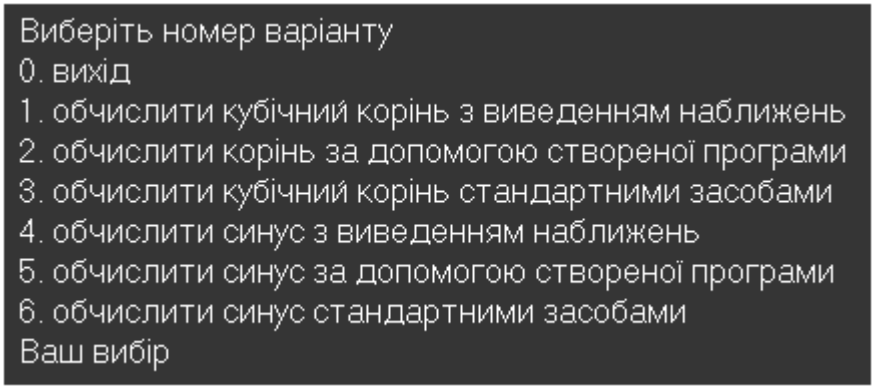


Рисунок 5.7 – Вигляд консолі після запуску програми

5.2.2 Допоміжні файли

На рисунку 5.8 наведено перелік файлів, які необхідно включити до програми, щоб забезпечити її нормальне функціонування.

```
#include <iostream>
#include <cmath>

using namespace std;
```

Рисунок 5.8 – Директиви #include програмного файлу

Файл iostream забезпечує реалізацію обміну інформацією з консоллю.

Файл cmath забезпечує доступ до математичних функцій.

5.2.3 Прототипи функцій

На рисунку 5.9 наведено перелік функцій, які будуть використовуватися у файлі програми.

Функція sqr3(double, double) забезпечує обчислення кубічного кореня із заданою точністю.

Функція sqr3_appr(double, double) окрім обчислення кубічного кореня забезпечує виведення на консоль послідовних наближень до дійсного значення кореня.

Функція our_sin(double, double) забезпечує обчислення синуса із заданою точністю.

Функція our_sin_appr(double, double) окрім обчислення синуса забезпечує виведення на консоль послідовності значень накопиченої суми членів нескінченного ряду.

Функція operate_chois(int) обробляє номер вибраного варіанту.

```
double sqr3(double a, double eps);
double sqr3_appr(double a, double eps);
double our_sin(double x, double eps);
double our_sin_appr(double x, double eps);
void operate_chois(int i);
```

Рисунок 5.9 – Прототипи функцій програмного файлу

5.2.4 Функція main() програмного файлу

Функція main() програмного файлу забезпечує виведення на консоль меню варіантів і початковий аналіз реакції користувача. У разі введення нуля програма закінчує роботу. В іншому випадку викликається функція operate_chois(int), яка забезпечує подальшу обробку вибраного номера варіанта. Після завершення обробки варіанту на консоль знову виводиться меню варіантів. Код цієї функції наведено на рисунку 5.10.

```
int main()
{
   int i=0;
   while(true){
      cout << "\nВиберіть номер варіанту\n";
      cout << "0.вихід\n";
      cout << "1. обчислити кубічний корінь з виведенням наближень\n";
      cout << "2. обчислити корінь за допомогою створеної програми\n";
      cout << "3. обчислити кубічний корінь стандартними засобами\n";
      cout << "4. обчислити синус з виведенням наближень\n";
      cout << "5. обчислити синус за допомогою створеної програми\n";
      cout << "6. обчислити синус стандартними засобами\n";
      cout << "Ваш вибір ";
      cin >> i;
      if(i==0)break;
      operate_chois(i);
   }
   return 0;
}
```

Рисунок 5.10 – Функція main() програмного файлу

5.2.5 Функція обробки номеру варіанта

Робота цієї функції починається з введення значень аргументу та необхідної точності обчислень для функції що підлягає тестуванню. Після цього за допомогою оператора swith обробляється варіант, що вибрав користувач. Код функції наведено на рисунку 5.11.

```
void operate_chois(int i){
    double x, eps;
    cout << "Задайте x = ";
    cin >>x;
    cout << "Задайте eps = ";
    cin >>eps;
    switch(i){
    case 1: sqr3_appr(x,eps);
    case 2: cout <<"обчислене значення кубічного кореня із "
                 << x <<" дорівнює " <<sqr3(x,eps)<<"\n";
    case 3: cout <<"стандартне значення кубічного кореня із "
                 << x <<" дорівнює " << pow(x,1.0/3)<<"\n";
        break;
    case 4: our_sin_appr(x,eps);
    case 5: cout <<"обчислене значення синуса для "<< x
                 <<" дорівнює " << our_sin(x,eps)<<"\n";
    case 6: cout <<"стандартне значення синуса для "<< x
                 <<" дорівнює " << sin(x)<<"\n";
        break;
    }
}
```

Рисунок 5.11 - Функція обробки номеру варіанта

5.2.6 Функції для обчислення кубічного кореня

На рисунку 5.12 наведено текст функції для обчислення кубічного кореня, яка реалізована відповідно до схеми алгоритму 5.5. Ця функція викликається у разі вибору варіанту за номером 2.

```
double sqr3(double a, double eps){
    double xOld, xNew = a;
    do{
        xOld = xNew;
        if(xOld == 0)break;
        xNew = xOld -(xOld*xOld*xOld-a)/(3*xOld*xOld);
    }while(abs(xNew - xOld) > eps);
    return xNew;
}
```

Рисунок 5.12 – Функція обчислення кубічного кореня

На рисунку 5.13 наведено текст тієї ж функції до якої добавлені засоби

для виведення послідовних наближень до значення кореня. Ця функція викликається у разі вибору варіанту за номером 1.

```
double sqr3_appr(double a, double eps){
    cout << "Обчислення кубічного кореня із "<< a <<"\n";
    cout << "послідовні наближення:\n";
    double xOld,xNew=a;
    do{
        xOld=xNew;
        if(xOld==0)break;
        xNew=xOld-(xOld*xOld*xOld-a)/(3.0*xOld*xOld);
        cout << xNew <<"\n";
    }while(abs(xNew-xOld)>eps);
    return xNew;
}
```

Рисунок 5.13 – Функція для обчислення кореня кубічного з виведенням послідовних наближень до кореня

5.2.7 Функції для обчислення синуса

На рисунку 5.14 наведено текст функції для обчислення синусу, яка реалізована відповідно до схеми алгоритму 5.6. Ця функція викликається у разі вибору варіанту за номером 5.

```
double our_sin(double x, double eps){
    int i=1; double u=x,result=u;
    while(abs(u)>eps){
        i++;
        u=-u*x*x/(2*i-2)/(2*i-1);
        result+=u;
    }
    return result;
}
```

Рисунок 5.14 – Функція для обчислення синуса

На рисунку 5.15 наведено текст тієї ж функції до якої добавлені засоби для виведення послідовних значень суми ряду. Ця функція викликається у разі вибору варіанту за номером 4.

```
double our_sin_appr(double x, double eps){
    int i=1; double u=x,result=u;
    cout << "Обчислення синуса для "<< x <<"\n";
    cout << "послідовні наближення:\n";
    while(abs(u)>eps){
        i++;
        u=-u*x*x/(2*i-2)/(2*i-1);
        result+=u;
        cout << result <<"\n";
    }
    return result;
}
```

Рисунок 5.11 – Функція для обчислення синуса з виведенням послідовних значень суми ряду

5.3 ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

У межах виконання лабораторної роботи студент повинен самостійно розробити алгоритми та програми розв'язання рівняння ітераційним методом у відповідності зі своїм варіантом завдання з таблиці 5.1 та обчислення суми безкінечного ряду відповідно варіанту з таблиці 5.2. Номер варіанта вибирається відповідно до останньої цифри залікової книжки.

5.4 ВИМОГИ ДО ЗВІТУ

- Назва роботи.
- Мета роботи.
- Короткий опис операторів **while** та **do … while**.
- Умови індивідуальних завдань та схеми алгоритмів їх вирішення.
- Тексти функцій для індивідуальних завдань з коментарями.
- Результати тестування проекту у вигляді копій консолі.
- Висновки.

5.5 КОНТРОЛЬНІ ПИТАННЯ

- Опис оператора **while**. Приклад використання.
- Опис оператора **do … while**. Приклад використання.
- Схема алгоритму обчислення кубічного кореня.
- Схема алгоритму обчислення синуса.
- Схеми алгоритмів вирішення індивідуальних завдань.
- Пояснення текстів функцій.
- Написати функцію обчислення кубічного кореня.
- Написати функцію обчислення синуса.
- Написати функцію для будь якого варіанту індивідуальних завдань.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программированя С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с
2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. –1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вільямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

6 ЛАБОРАТОРНАЯ РАБОТА № 6. ОБРОБКА ДАННЫХ ЗА ДОПОМОГОЮ ЦИКЛУ FOR

Мета роботи:
- Ознайомитися з оператором циклу **for**.
- Ознайомитися з функцією генерації випадкових чисел.
- Написати програми, що забезпечують обработку данных за допомогою операторів циклу **for**.

6.1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІІ

6.1.1 Оператор циклу for

Оператор **for** у мові С - це дуже потужний інструмент для організації циклів. За його допомогою можна запрограмувати більшість циклічних процесів, але він не такий прозорий, як оператор **while.**

У загальному вигляді оператор **for** записується таким чином:

```
for (<вираз1>; <вираз2>; <вираз3>)
        <тіло циклу>;
```

Рисунок 6.1– Синтаксис оператора **for**

На цьому рисунку < вираз1> - це вираз ініціалізації, який встановлює початкові значення змінних циклу; <вираз2> - це вираз умови, що задає умову виконання тіла циклу; <вираз3> - це вираз ітерації, який виконує зміну значень змінних циклу; <тіло циклу> - це оператор або блок, який задає дії, що мають повторюватися.

Оператор циклу **for** виконується наступним чином:

1) Обчислюється < вираз1>, внаслідок чого змінні циклу приймають початкові значення. Цей вираз обчислюється тільки один раз на початку роботи циклу.
2) Обчислюється <вираз2>, що є умовою продовження виконання циклу.
3) Якщо умова хибна, то цикл закінчується.
4) Виконується тіло циклу.
5) Обчислюється < вираз3>, який використовують для зміни параметрів циклу.
6) Виконання продовжується з пункту 2.

Параметри циклу можна оголошувати як у виразі1, так і за межами циклу. Але у першому випадку параметри циклу будуть доступні тільки в межах циклу

Як приклад використання циклу **for** розглянемо функцію обчислення числа Фібоначчі із заданим номером. Це послідовність чисел у якій нульове число дорівнює 0 і перше дорівнює 1. Решта чисел обчислюється як сума двох

попередніх. Тобто послідовність має такий вигляд 0 1 1 2 3 5 8 13 21 34 …

У цій програмі у тілі циклу послідовно обчислюються числа Фібоначчі починаючи з другого і до заданого, з номером n.

```
uint fibo(uint n){
   if(n==0 || n==1) return n;
   uint  f, f0=0, f1=1; //Початкові значення двох попередніх чисел
   for(uint i = 2; i<=n; i++){
       f=f0+f1; //Обчислюємо число Фібоначчі
       f0=f1;   f1=f;  // Змінюємо значення двох попередніх чисел
    }
   return f;
}
```

У наступному прикладі цикл for використовується для підрахунку факторіала числа n.

```
unsigned long long fact(uint n) {
   unsigned long long f=1;
   for( uint i = 1; i<=n; i++ )
      f*=i;
   return f;
}
```

6.1.2 Особливості використання циклу for

У попередньому пункті були розглянуті приклади використання циклу **for** де використовувався усього один параметр циклу, усі три вирази і тіло циклу. Але можливі і інші варіанти використання цього циклу.

Так, наявність кожного з трьох виразів заголовку циклу не обов'язкова. Може навіть не бути жодного з них. Таким чином отримуємо безкінечний цикл. Наприклад, розглянуту вище функцію обчислення факторіалу можна записати і таким чином:

```
unsigned long long fact(uint n) {
   unsigned long long f=1;
   uint i = 1;
   for( ; ; ){
      f*=i;
      if(i>=n) return f;
      i++;
   }
}
```

Не важко здогадатися, що коли цикл має тільки вираз перевірки умови, то він буде еквівалентом циклу while.

У програмі підрахунку суми цифр цілого числа, що була розглянута у попередній лабораторній роботі, цикл while можна замінити циклом for і програма буде виглядати так:

```
cout<<"/nВведіть ціле число\n ";
int x;
cin>>x;
int sum=0;
for( ; x!=0; ){
   sum+=x%10;
   x/=10;
}
cout<<"/nСума цифр у числі дорівнює "<<sum<<"\n ";
```

Ще одна особливість циклу **for** пов'язана з операцією кома «,». Ця операція пов'язує декілька виразів, що розглядаються як один. Результатом такої послідовності буде результат обчислення останнього виразу.

Можливість використання операції кома дозволяє використовувати декілька параметрів циклу, а використання цієї операції у ітераційному виразі може призвести до зникнення тіла циклу.

Ось як, наприклад може виглядати функція обчислення факторіалу:

```
unsigned long long fact1(uint n) {
   unsigned long long f=1;
   for( uint i = 1; i<=n; f*=i, i++ );
   return f;
}
```

А так може виглядати функція обчислення числа Фібоначчі:

```
int fibo1(unsigned int n){
   if(n==0 || n==1) return n;
   unsigned int f=0;
   for(int i=2,f0=0,f1=1; i<=n; f=f1+f0, f0=f1, f1=f, i++);
   return f;
}
```

6.1.3 Випадкові числа

Для формування випадкових чисел в Qt використовується функція qrand(), яка повертає псевдо випадкове число з діапазону 0.. RAND_MAX-1.

RAND_MAX=0x7FFF. Числа вважаються рівномірно розподіленими у цьому діапазоні. Кожний наступний виклик цієї функції призводить до генерування наступного випадкового числа, яке пов’язано з попереднім. Саме тому числа називають псевдо випадковими.

Функцію qsrand() використовують для завдання початкового числа (зерна) псевдо випадкової послідовності. В якості аргументу у цю функцію слід передати ціле число. Якщо цю функцію не використовувати, то функція qrand() буде завжди формувати ту саму послідовність.

Для того, щоб початкове число було випадковим, можна для його формування використовувати логічну комбінацію значення поточного часу у секундах і значення часу процесора від початку виконання програми у так званих тіках. Тік зазвичай дорівнює одній мілісекунді, а системний час відраховується від 1 січня 1970 року за Гринвічем.

Виклик функції qsrand() може виглядати так:

```
qsrand(time(NULL) | clock());
```

Для того, щоб отримати цілі числа із обмеженого діапазону, наприклад від 0 до N-1, можна використовувати операцію %.

```
int x = qrand() % N;
```

Для того, щоб отримати цілі числа із діапазону, від -N до N-1, можна використовувати такий вираз:

```
int x = qrand() % (2*N) - N;
```

Якщо потрібні дійсні випадкові числа з діапазону від 0 до 1 можна використовувати такий вираз:

```
float x = qrand() / (float) RAND_MAX;
```

6.1.4 Табулювання функцій

Табулювання значень деякої функції є досить поширеним завданням, де використовуються цикли. Суть завдання полягає в тому, що необхідно змінювати аргумент функції від деякого початкового значення до кінцевого значення з деяким кроком і для отриманих значень аргументу обчислювати значення функції. Цю задачу можна вирішувати за допомогою циклу for декількома способами. На перший погляд найбільш природно у циклі for параметр циклу послідовно збільшувати на величину кроку. Однак такий спосіб вважається не зовсім правильним. Зазвичай аргументи функцій - дійсні числа, які в пам'яті машини записуються з деякою похибкою. Багаторазове підсумовуванні таких чисел призводить до накопичення похибки, яка може стати помітною при великій кількості циклів. Тому для табулювання функцій зручніше використовувати цикл for, в якому параметром є цілочисловий номер рядка таблиці. Рядки таблиці зручно вважати пронумерованим з 0, тоді прирощення аргументу на кожному кроці можна визначати шляхом множення кроку на номер рядка. Помилка при цьому не накопичується.

6.2 ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

Створити проект, який би забезпечив виконання трьох підпрограм з використанням циклу for.

Перша підпрограма має забезпечити табулювання функції відповідно до варіанту з таблиці 3.1 (лабораторна робота №3) і використовувати функцію розрахунку за формулою, що була створена у тій лабораторній роботі.

Друга підпрограма має забезпечити обробку цілих чисел у відповідності з завданням з таблиці 6.1, обраним відповідно до останньої цифри номера залікової книжки.

Таблиця 6.1 – Варіанти завдань з обробки цілих чисел

Варіант	Завдання
1	2
0	Написати програму, що буде виводити на консоль у зростаючому порядку і підраховувати тризначні цілі числа, в яких немає однакових цифр. Слід попередньо написати функцію, що визначає, чи є однакові цифри в числі.
1	Написати програму, що буде виводити на консоль у зворотному порядку і підраховувати тризначні цілі числа, в яких є поспіль дві однакові цифри. Попередньо написати функцію, що визначає, чи є в числі поспіль дві однакові цифри.
2	Написати програму, що буде виводити на консоль тризначні цілі числа, сума цифр яких дорівнює введеному числу N (1<N<27). Визначити кількість цих чисел. Попередньо написати функцію, яка обчислює суму цифр числа.
3	Написати програму, що буде виводити на консоль всі прості числа, які менші за N, і підраховувати їх кількість. Попередньо написати функцію тестування числа на простоту. Простим називають ціле число, яке без остачі ділиться тільки на 1 і на себе. Ознакою є наявність залишку від ділення числа X на числа від 2 до X / 2.
4	Написати програму, що буде виводити на консоль цілі числа з інтервалу від n1 до n2 і суми дільників (за винятком 1 і самого числа) для кожного з них. Для знаходження дільників слід перевіряти результат ділення числа N на числа від 2 до N / 2. Попередньо написати функцію підрахунку суми дільників одного числа.
5	Написати програму, що буде виводити на консоль цілі числа з інтервалу від n1 до n2 до складу яких входить задана цифра. . Попередньо написати функцію, що визначає, чи є в числі потрібна цифра.
6	Написати програму, що буде знаходити мінімальне значення функції $f(x)=10x^2+215x+100$ на інтервалі цілих чисел від n1 до n2. Попередньо написати функцію, що обчислює значення f(x).

Продовження таблиці 6.1

1	2
7	Написати програму, що буде виводити на консоль цілі числа з інтервалу від n1 до n2 і перелік їх простих дільників (за винятком 1 і самого числа). Для знаходження дільників слід перевіряти результат ділення числа N на числа від 2 до N / 2. Попередньо написати функцію пошуку дільників одного числа.
8	Написати програму, що буде виводити на консоль значення факторіалу цілих чисел з інтервалу від n1 до n2. Попередньо написати функцію, що обчислює факторіал числа.
9	Написати програму, що буде виводити на консоль значення числа Фібоначчі з номерами від n1 до n2. Попередньо написати функцію, що обчислює число Фібоначчі із заданим номером.

Третя підпрограма має забезпечити обчислення характеристики послідовності випадкових чисел за модулем 100, відповідно до варіанту з таблиці 6.2. Характеристика має бути обчислена 10 разів поспіль і кожен з результатів слід вивести на консоль. Довжину послідовності випадкових чисел має задавати користувач.

Таблиця 6.2 – Варіанти завдань з обробки послідовностей випадкових чисел

Варіант	Характеристика послідовності, що має бути обчислена
0	Максимальне та мінімальне число
1	Найбільша довжина послідовності, числа якої зменшуються
2	Середнє значення
3	Дисперсія $(\sum(x^2))/n - ((\sum(x))/n)^2$
4	Кількість повторень заданого числа
5	Кількість парних чисел
6	Кількість непарних чисел
7	Різниця між максимальним та мінімальним числом (розмах)
8	Кількість чисел кратних 5
9	Номер заданого числа у послідовності

6.3 ПРИКЛАД СТВОРЕННЯ ПРОЕКТУ «ЦИКЛ FOR»

У цій лабораторній роботі ми створимо консольний додаток, який продемонструє можливості циклу для обробки різних даних. До програми додамо такі допоміжні файли, рисунок 6.2.

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

Рисунок 6.2 – Допоміжні файли застосування

Файл iostream забезпечує реалізацію обміну інформацією з консоллю.
Файл cmath забезпечує доступ до математичних функцій.
Файл windows.h дає доступ до системних утиліт windows.
Файл cmath забезпечує доступ до функцій часу.

6.3.1 Головна функція проекту

Головна функція проекту має надати можливість користувачеві викликати потрібну програму з використанням циклу for.

Вигляд основного фрагменту функції main() наведено на рисунку 6.3.

```
char ch;
do{
   system("cls");
   SetConsoleTitleA("Використання циклу for");
   cout<<"Варіанти використання циклу for:\n";
   cout<<"1.Табулювання функції\n";
   cout<<"2.Обробка послідовності цілих чисел\n";
   cout<<"3.Обробка послідовності випадкових чисел\n";
   cout<<"Виберіть номер варіанту ";
   cin>>ch;
   switch(ch){
      case '1':tabFunc();break;
      case '2':intChain();break;
      case '3':rndChain();break;
   }
   cout<<"\nДля продовження введіть *";
   cin>>ch;
}while(true);
```

Рисунок 6.3 – Основна частина функції main()

6.3.2 Табулювання функції

У якості прикладу розглянемо наведену нижче, на рисунку 6.4, підпрограму табулювання функції y = a*e $^{(-b*x)}$ *sin(w*x). У програмі використано рекомендації пункту 6.1.4.

Результати роботи програми наведено на рисунку 6.5.

```
void tabFunc(){
   system("cls");
   SetConsoleTitleA("Табулювання функції y=a*exp(-b*x)*sin(w*x)");
   cout<< "Введіть коефіцієнти формули a, b, w\n";
   float a,b,w; cin>>a>>b>>w;
   cout<< "Введіть початкове і кінечне значення аргументу\n";
   float x0,  xMax; cin >>x0>>xMax;
   cout<< "Введіть крок зміни аргументу\n";
   float step; cin >> step;
    // Оголошуємо внутрішні змінні
    float x;  //поточне значення аргументу
    float y; // поточне значення  функції
    int k; // Номер останнього рядка у таблиці
    // Округлюємо кількість рядків
    k = ceil((xMax-x0)/step);
    for(int i=0; i<=k;i++){
       x = x0 + step * i; // чергове значення аргументу
       y = a*exp(-b*x)*sin(w*x);
       cout.width(5); cout.precision(3); cout<<x;
       cout.width(15); cout.precision(4); cout<<y<<"\n";
    }
}
```

Рисунок 6.4 – Програма виведення таблиці значень функції

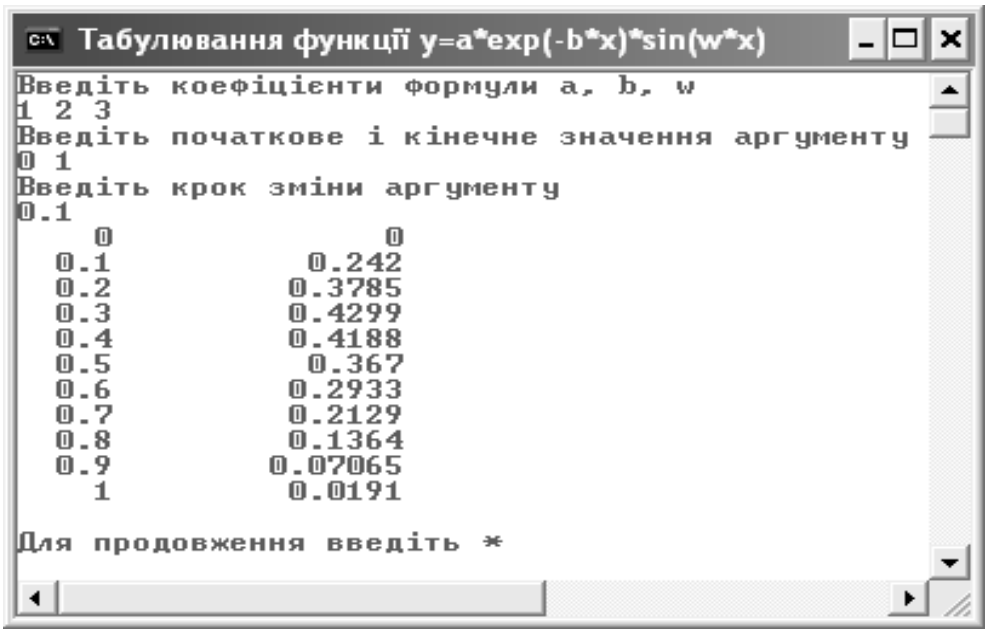


Рисунок 6.5 – Результати табулювання значень функції

6.3.3 *Обробка послідовностей цілих чисел*

Як приклад розглянемо таку задачу: «В послідовності цілих чисел від 1 до N потрібно виділити числа, квадрат яких дорівнює сумі квадратів інших цілих чисел».

Для вирішення цієї задачі спочатку напишемо функцію, що з'ясовує, чи можна представити квадрат одного числа, як суму квадратів інших цілих чисел. Після цього можна буде викликати цю функцію для усіх чисел із потрібного інтервалу.

6.3.3.1 Розробка допоміжної функції

Функцію, що з'ясовує, чи можна представити квадрат одного числа, як суму квадратів інших цілих чисел назвемо hasSqr. Хай вона повертає тип bool. Якщо буде знайдена пара чисел, сума квадратів яких дорівнює квадрату заданого числу, функція має повертати true, у іншому випадку має повертати false.

Самі числа, квадрати яких утворюють квадрат заданого числа, отримаємо через параметри функції, для чого ці параметри оголосимо такими, що передаються через посилання. А першим параметром функції зробимо число, що підлягає тестуванню. Таким чином прототип функції буде таким:

```
bool hasSqr(uint x, uint &x1, uint &x2);
```

Тепер можна перейти до реалізації функції. Задачу будемо вирішувати шляхом аналізу усіх комбінацій чисел x1 та x2. Перше число x1 будемо змінювати від 1 до x-1. А для кожного значення x1 будемо аналізувати значення x2 від x1 до x2-1. Перебір значень x2 починаємо від поточного значення x1 для того, щоб уникнути аналізу дзеркальних комбінацій, наприклад 3 і 5 та 5 і 3.

У тому випадку, якщо комбінація знайдена, робота функції закінчується. Тобто функція не знаходить усі можливі комбінації, а повертає першу ліпшу знайдену. Нижче наведено текст цієї функції.

```
bool hasSqr(uint x, uint &x1, uint &x2){
   for( x1 = 1; x1 < x;  x1++) {
      for( x2 = x1; x2 < x; x2++)
         if( x*x == x1*x1+x2*x2)
            return true;
   }
   return false;
}
```

6.3.3.2 Розробка функції для вирішення головної задачі

Маючи функцію hasSqr() можемо перейти до вирішення головної задачі. Для цього достатньо за допомогою циклу for організувати перебір усіх чисел із заданого діапазону і викликати для кожного з них функцію hasSqr(). Якщо функція поверне true, то слід вивести на консоль значення параметрів функції. Нижче наведено текст функції для вирішення головної задачі.

```
void  intChain(){
   cout<<"Введіть число n1>1 та число n2>n1 ";
   uint x1, x2, a, b;
   cin>>x1>>x2;
   for(int i=x1; i<=x2; i++){
      if(hasSqr(i, a, b)){
         cout.width(5);
         cout<<i<<"^2="<<a<<"^2+"<<b<<"^2"<<"\n";
      }
   }
}
```

Нагадаємо, що змінні a та b отримують значення у функції hasSqr, тому що вони передаються у функцію через посилання.

Результат виконання цієї програми маємо на рисунку 6.6.

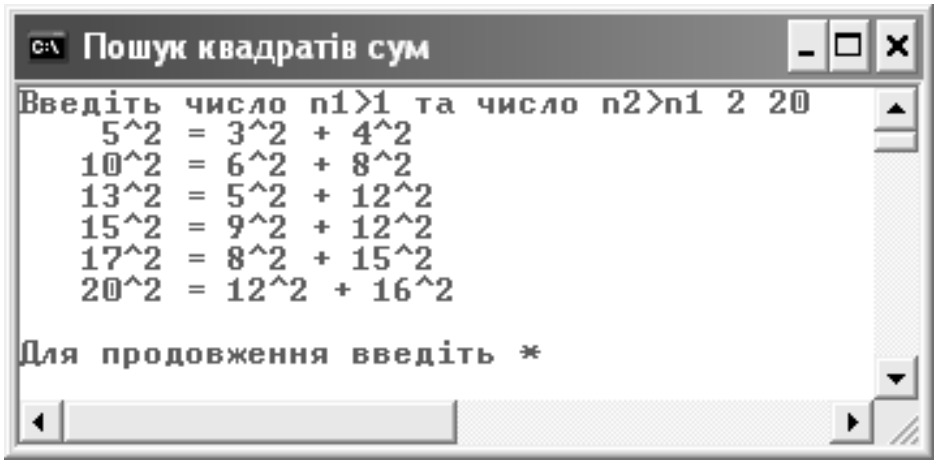


Рисунок 6.6 – Результат пошуку пар чисел, сума квадратів яких дорівнює квадрату заданого числу

6.3.4 Обробка послідовності випадкових чисел

Як приклад обробки послідовності випадкових чисел, розглянемо задачу визначення максимальної довжини зростаючої послідовності чисел у випадковій послідовності чисел, що приймають значення від 0 до 99. Програма повинна забезпечити пошук максимальної довжини послідовності 10 разів поспіль і виводити результати на консоль.

Для отримання випадкової послідовності будемо використовувати функцію qrand() із урахуванням рекомендацій пункту 6.1.3.

У програмі використовується два цикли for. Перший забезпечує десяти разовий повтор обробки послідовності випадкових чисел. Другий, внутрішній цикл забезпечує формування послідовності заданого розміру і пошук найбільшої довжини зростаючої послідовності. Текст програми наведено на рисунку 6.6. Результати виконання програми на рисунку 6.7.

```
void rndChain(){
   system("cls");
   SetConsoleTitleA("Довжина зростаючої послідовності");
   qsrand(time(NULL)|clock());
   uint n;
   cout<<"Введіть обсяг вибірки ";
   cin>>n;
   for(int i=0;i<10;i++){
      uint maxLen=0, len=0, pred =100;
      for(int j=0;j<n;j++){
         int x=qrand()%100;
         if(x>pred){
            len++;
            if(len>maxLen)maxLen=len;
         }
         else  len=1;
         pred=x;
      }
      cout<<"Максимальна довжина послідовності дорівнює "<<maxLen<<"\n";
   }
}
```

Рисунок 6.6 – Функція для визначення максимальної довжини зростаючої послідовності у послідовності випадкових чисел

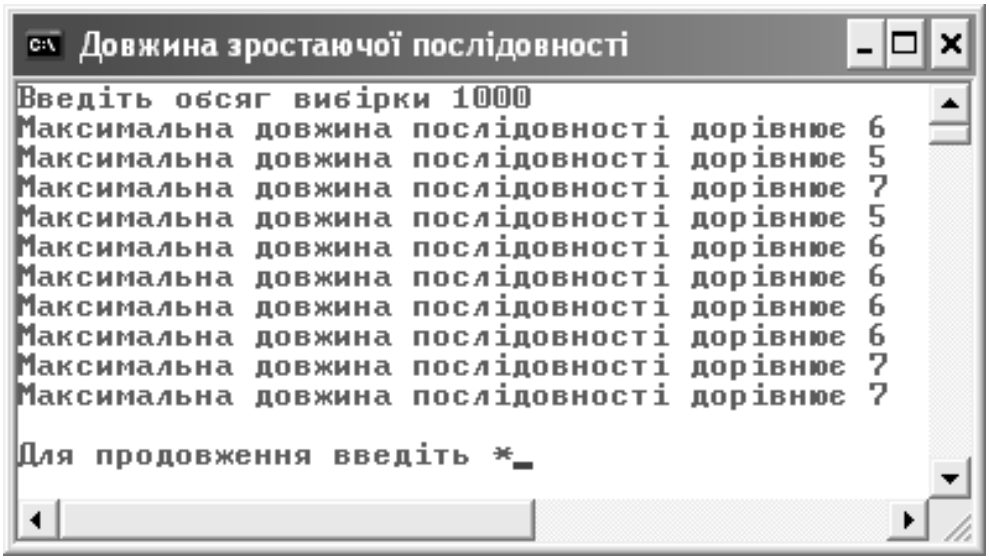


Рисунок 6.7 – Результати визначення максимальної довжини зростаючої послідовності у послідовності випадкових чисел

6.4 ВИМОГИ ДО ЗВІТУ

- Назва роботи.
- Мета роботи.

– Короткий опис оператора **for**.
– Особливості використання оператора **for.**
– Генерація випадкових чисел.
– Тексти функцій для індивідуальних завдань з коментарями.
– Результати тестування проекту у вигляді копій консолі.
– Висновки.

6.5 КОНТРОЛЬНІ ПИТАННЯ

– Опис оператора **for**. Приклад використання.
– Приклад використання оператора **for** з операцією кома.
– Особливості використання функції qrand().
– Написати функцію або програму за вказівкою викладача з використанням циклу for.
– Написать функцію або програму відповідно до одного із варіантів індивідуальних завдань до лабораторної роботи.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программированя С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с
2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. –1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вильямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

7 ЛАБОРАТОРНА РОБОТА № 7. ОДНОВИМІРНІ МАСИВИ

Мета роботи:
- Ознайомитися з поняттями масив.
- Навчитися оголошувати та ініціалізувати одновимірні масиви.
- Ознайомитися з особливостями масивів символів.
- Навчитися оперувати масивами, як параметрами функцій.
- Познайомитися з операціями над масивами.
- Створити програму для обробки масивів.

7.1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО МАСИВИ

Масив являє собою сукупність даних, що організована певним чином, тобто структуру даних. Основні особливості структури даних, що зветься масивом полягають у наступному:

– масив складається з елементів, які мають однаковий тип;

– елементи масиву послідовно розташовані в одній ділянці оперативної пам’яті без проміжків між елементами;

– кожен з елементів масиву має свій порядковий номер, що зветься індексом;

– нумерація елементів починається з 0;

– до елементів масиву можна звертатися використовуючи ім’я масиву і індекс;.

– масив може бути одновимірним, або багатовимірним, тобто таким у якого кожний елемент є також масивом;

– у мові С, С++ ім’я масиву зберігає адресу першого елементу цього масиву, тобто є вказівником на його початок.

7.1.1 *Оголошення одновимірного масиву та звернення до його елементів*

Оголошення одновимірного масиву має вигляд, представлений на рисунку 7.1.

<тип елементів> <ім’я масиву> [<кількість елементів>] ;

Рисунок 7.1 – Синтаксис оголошення одновимірного масиву

Тип елементів визначає тип елементів, з яких складається масив. Це може бути будь який допустимий у мові тип, простий або складений

Ім’я масиву – це ідентифікатор написаний за правилами запису імен у мові С С++

Кількість елементів – це константа, або константний вираз, що визначає розмір даного масиву.

Приклад оголошення масиву з ім'ям А, що складається з десяти елементів цілого типу наведено нижче:

```
int  A [10] ;
```

Як і у випадках із оголошенням простих змінних, під час оголошення масиву можна ініціалізувати його усі елементи або тільки декілька початкових. Приклад оголошення масиву з ініціалізацією трьох елементів із десяти наведено нижче:

```
int  A [10] ={2,5,10};
```

Незалежно від того, скільки елементів масиву було ініціалізовано при оголошенні, пам’ять виділяється під усі елементи. Значення елементів масиву, що не були ініціалізовані, невизначені («сміття») або дорівнюють нулю, якщо масив визначений як глобальний чи статичний.

Якщо у оголошенні масиву ініціалізуються усі елементи (повна ініціалізація), то кількість елементів у оголошенні можна не показувати, хоча квадратні дужки залишаються. Приклад такого оголошення наведено нижче:

```
int  A [] ={2, 5, 10, 3, 6, 0, 9, 4, 5, 7};
```

Проініціалізований таким чином масив, для наочності представимо рисунком 7.2.

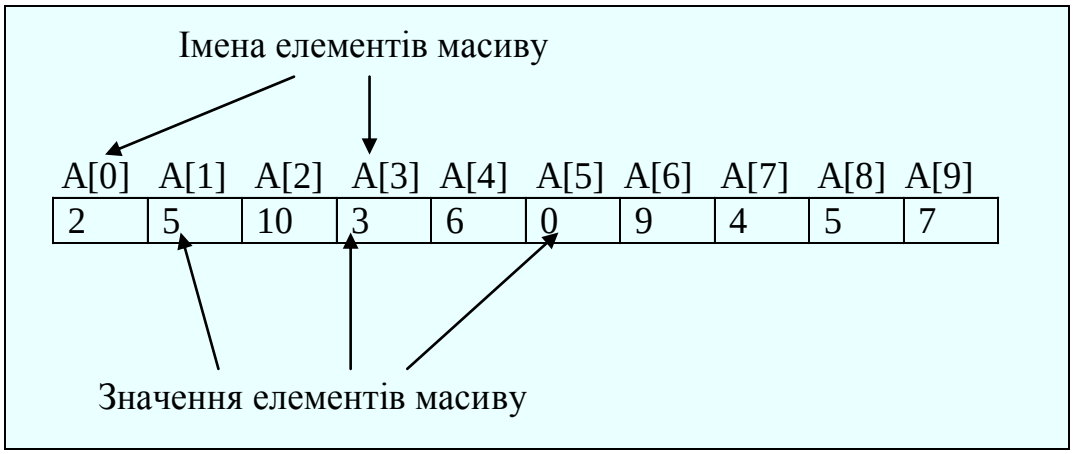


Рисунок 7.2 – Схема одновимірного масиву

Для доступу до елементів масиву використовується синтаксична конструкція, що складається з імені масиву та індексу, який записується у квадратних дужках. Тобто доступ до елементів цього масиву забезпечується виразом: A[i]. Індекс і в даному прикладі є цілим числом у діапазоні від 0 до 9. Таким чином, А [0] - це ім'я першого елемента, і т.д., А [9] - ім'я останнього елементу.

Індексовані елементи масиву можуть бути використані так само, як і прості змінні. Наприклад, вони можуть перебувати у виразах як операнди, їм можна привласнювати будь-які значення, відповідні їх типу.

Працюючи з масивами у програмах на мові С слід пам’ятати, що ніякого

контролю за значеннями індексів, що використовуються для доступу до елементів масиву нема. Ви можете звернутися до «елементу масиву» з будь яким номером, але отримаєте невідомо що. Ще гірше, якщо ви зміните значення цього «елементу масиву». Це може призвести до катастрофічних наслідків для вашої програми.

7.1.2 *Приклад використання одновимірного масиву*

Нижче наведено приклад створення масиву для збереження чисел Фібоначчі. Два перших числа Фібоначчі рівні 0 та 1, а кожне наступне дорівнює сумі двох попередніх. У програмі перші 2 елементи масиву заповнюються до циклу, решта - в циклі.

```
int a[10];
   a[0] = 0;
   a[1] = 1;
   for (i=2 ; i<10; i++)
      a[i] = a[i – 2] + a[ i – 1 ];
```

7.1.3 *Масиви символів*

У мові С не було спеціального типу для оголошення рядків символів, у С++ і Qt такі типи є, але ми не будемо поки що їх розглядати.

Коли у тексті програми на мові С з'являлася константа, охоплена подвійними лапками, наприклад, "Невже це не рядок символів?", то вона розглядалася, як масив. Для такого масиву в оперативній пам'яті виділялася ділянка, розмір якої був на один байт більший, ніж кількість символів у рядку. Цей додатковий байт і досі використовується для збереження ознаки кінця рядка. У якості такої ознаки використовується символ '\0'.

Символьні рядки, які є змінними програми, оголошуються, як звичайні масиви, рисунок 7.3.

```
char   <ім'я символьного рядка> [<кількість байтів >] ;
```

Рисунок 7.3 – Синтаксис оголошення рядка символів

Приклад оголошення рядка символів з ім'ям str, для якого у памяті буде виділено 100 байтів наведено нижче:

```
char  str[100] ;
```

Як і у випадках із оголошенням масивів чисел, масиви символів можна ініціалізувати. Незалежно від того, скільки символів було ініціалізовано при оголошенні, пам'ять виділяється під усі елементи. Значення символів, що не були ініціалізовані, невизначені («сміття») або дорівнюють нулю, якщо масив визначений як глобальний чи статичний.

Приклади оголошення рядків символів з одночасною ініціалізацією наведено нижче:

```
char  str[10] ={‘H’, ‘e’, ‘l’, ‘l’, ‘o’,  ‘!’, ‘\0’} ;
char  str[10] =”Hello!”;
```

Зверніть увагу, якщо рядок ініціалізується як класичний масив, з окремих символів, то слід не забути про ознаку кінця рядка. Та на щастя масив символів можна ініціалізувати за допомогою рядка символів у подвійних лапках. Ознака кінця рядка у цьому випадку додається автоматично.

Якщо у оголошенні рядка символів ініціалізуються усі елементи (повна ініціалізація), то кількість елементів у оголошенні можна не показувати, хоча квадратні дужки залишаються. Приклад такого оголошення наведено нижче:

```
    char  str[] =”Hello!”;
```

Що стосується обробки рядків (масивів символів) то усе залишається таким самим, як і у випадку числових масивів.

7.1.4 *Одновимірні масиви як параметри функцій*

Якщо формальним параметром функції є масив, то він оголошується майже так само, як і прості змінні, лише після його імені слід поставити пусті квадратні дужки. Тип масиву при цьому записується таким же як і тип елементів масиву.

Слід також пам’ятати, що хоча перед іменем масиву, як формального параметру, символ & не ставиться, та все одно масив буде передано за посиланням. А символ & ставити не потрібно тому, що ім’я масиву і є адресою першого елементу масиву.

Крім того, слід брати до уваги той факт, що масив «не знає», скільки у нього елементів, тому передаючи масив до функції слід передавати і кількість елементів масиву, що має бути оброблена. Це число, звичайно, не може перевищувати кількість елементів під які виділено пам’ять під час оголошення масиву.

Як приклад розглянемо функцію, що знаходить і повертає максимальний елемент масиву:

```
int max(int m[], int n){
   int mx=INT_MIN;
   for(int i=0; i<n; i++)
      if (m[i]>mx) mx=m[i];
   return mx;
}
```

Слід звернути увагу на те, що у мові С С++ такого типу як «масив», не існує, не можна написати int[]. З цієї причини у функції не можна вказати масив, як тип того, що повертається функцією, але масив можна повернути

через параметри функції. Ось, наприклад, як може виглядати функція, що повертає масив чисел Фібоніччі.

```
void fibo(int a[], int size){
        a[0] = 0;
        a[1] = 1;
        for (i=2 ; i<size; i++)
           a[i] = a[i – 2] + a[ i – 1 ];
}
```

Як бачимо, тип значення, що повертається тут void, але завдяки тому, що масиви передаються через посилання, функція заповнює числами Фібоначчі масив, посилання на який їй передано через параметр.

Щоправда, слід мати на увазі, що функція може повертати покажчик на масив, і таким чином проблема повернення масиву теж вирішується. Але покажчики ми поки що не розглядаємо. Ця тема буде розглянута трохи пізніше.

7.2 ФУНКЦІЇ ОБРОБКИ МАСИВІВ ЧИСЕЛ

Під час роботи з масивами чисел доводиться виконувати ряд специфічних операцій, пов'язаних саме з цією структурою даних числового типу. Найбільш поширеними з них є:

- введення масиву чисел;
- виведення масиву чисел;
- формування масиву випадкових чисел;
- пошук суми елементів масиву;
- пошук максимального та мінімального елементів масиву та їх індексів;
- пошук індексу елементу масиву за його значенням;
- циклічний зсув елементів масиву праворуч або ліворуч;
- видалення елементу з масиву;
- формування масиву накопичених значень елементів.

Нижче ми розглянемо деякі з таких функцій на прикладах обробки цілочислових масивів.

Незважаючи на відмінність завдань, що вирішуються цими функціями, у них буде дві однакові особливості.

Перша полягає в тому, що в кожну з цих функцій буде передаватися ім'я масиву і функція буде обробляти саме цей масив, а не його копію.

Друга особливість полягає у тому, що окрім масиву до функції слід передавати кількість даних у масиві, бо оголошений розмір масиву зазвичай перевищує кількість даних у ньому.

7.2.1 *Функція формування випадкового масиву*

Ця функція дещо відрізняється від функції генерації послідовності випадкових чисел, що була розглянута раніше. Різниця полягає у тому, що

сформовані числа записуються у елементи масиву. Текст функції наведено на рисунку 7.4.

```
void createRandomArray(int ar[], int size, int modul){
   for(int i=0; i<size; i++)
      ar[i] = rand()%modul;
};
```

Рисунок 7.4 – Функція формування випадкового масиву

7.2.2 *Функції виведення масиву на консоль*

Ця функція досить проста і не потребує коментарів. Ця функція досить проста і не потребує коментарів. Текст функції наведено на рисунку 7.5.

```
void arToConsole(int ar[], int size){
   for(int i=0; i<size;i++){
      cout<<ar[i];
      if(i<size-1)cout<<" ";
   }
   cout<<endl;
}
```

Рисунок 7.5 – Функція виведення масиву на консоль

7.2.3 *Функції введення масиву з консолі*

Можливі декілька варіантів введення масиву з консолі. Перший полягає у тому, що користувач одразу набирає числа масиву, розділяючи їх пробілами, тобто масив чисел являє собою рядок символів. Цей рядок вводиться за допомогою функції gets(), а далі перетворюється у послідовність чисел. Перевага цього способу полягає у простоті вводу, а недолік у тому, що доводиться перетворювати символьний рядок у масив чисел.

Другий варіант полягає у послідовному введенні розміру та кожного елементу масиву з консолі як чисел за допомогою об’єкту cin. Тут вже не потрібно турбуватися про перетворення символів у числа, але сама процедура введення ускладнена.

Нижче ми розглянемо обидва варіанти.

7.2.3.1 Функція введення масиву як рядка символів

Сама функція, що наведена нижче, виглядає досить просто і фактично забезпечує тільки введення рядка символів, а для перетворення рядка символів у масив використовується функція strToArr().

```
void getArFromConsole1(int ar[], int &size){
char str[80];
cout<<"Введіть елементи масиву через пробіли"<<endl;
gets(str);
strToArr(str, ar, size);
}
```

Рисунок 7.6 – Функція введення масиву як рядка символів

Функцію перетворення рядка символів у масив наведено на рисунку 7.7.

```
void strToArr(char str[], int ar[], int &size){
  int i (0);  size=0;
  for( ; ; ){
    while( str[i]!='\0'&&  str[i]==' ')i++;//ігноруємо пробіли
    if(str[i]=='\0')return;
    ar[size]=0;//Формуємо наступне число
    while(str[i]!='\0' && str[i]!=' '){
      if(!isdigit(str[i] )){
        cout<<str[i]<<" не цифра! \n";
        return;
      }
      ar[size]=ar[size]*10+(str[i]-48); i++;
    }
    size++;
  }
}
```

Рисунок 7.7 – Функція перетворення рядка символів у масив

У цій функції довжину рядка визначено розміром 80 виходячи з ширини консолі. Для введення рядка використовується функція gets() а не об'єкт cin, тому що цей об'єкт буде сприймати пробіл, як кінець рядка. Використання функції gets() потребує підключення заголовного файлу <cstdio> або < stdio.h>.

Роботу по перетворенню рядка символів у масив виконує функція strToArr():

У цій функції для перевірки, чи є символ цифрою, використовується функція isdigit(), що визначена у заголовному файлі <cctype> або <ctype.h>.

Для перетворення символу цифри у число використовується тай факт, що код цифри на 48 більший ніж значення цієї цифри.

Значення усього числа накопичується за схемою Горнера, відповідно до якої, наприклад, число 12345 обчислюється так: (((1*10+2)*10+3)*10+4)*10+5 .

7.2.4 *Функція введення масиву по елементам*

Текст функції наведено на рисунку 7.8.

```
void getArFromConsole2(int ar[], int &size) {
   cout<<"\n Введіть кількість елементів масиву ";
   cin>>size;
   for( int i=0; i<=size; i++){
      system("cls");
      cout<<" Усього кількість елементів масиву "<<size<<"\n";
      if (i>0){
         cout<<" Введені елементи масиву: \n";
         for(int j=0; j<i; j++){ cout<<ar[j]<<" "; }
         cout<<"\n";
      }
      if(i<size){
         cout<<"Введіть елемент масиву № "<<i<<": ";
         cin>>ar[i];
      }
   }
}
```

Рисунок 7.8 – Функція введення масиву по елементам

У цій функції кожен елемент масиву вводиться за допомогою об’єкту cin у циклі for. До початку цього циклу слід ввести кількість елементів масиву.

Функція могла б виглядати так само просто як і функція виведення масиву, що наведена у пункті 7.2.1, і ви можете її саме так і написати, але у наведеній нижче реалізації додано операції очищення екрану і виведення елементів, що були вже введені, і очищення екрану для того, щоб кожен елемент масиву вводився на тому ж самому місці, що й попередній.

Для того, щоб функція працювала, необхідно до складу проекту додати директиву включення заголовний файлу <windows.h> .

7.2.5 *Функція вилучення елементу з масиву*

У цій функції елементи масиву перебираються, поки не буде знайдено заданий елемент. Після цього усі наступні елементи зсуваються вліво на одну позицію, таким чином займаючи місце елементу, що видаляється. Розмір масиву зменшується на 1. Після цього пошук триває, поки не буде досягнутий кінець масиву.

Для перебору елементів масиву у функції використовується цикл **while,** тому що індекс елементу у разі зсуву елементів ліворуч не потрібно змінювати. Текст функції наведемо на рисунку 7.9.

```
void delElement(int element, int ar[], int &size){
  int i=0;
  while(i < size){
    if(ar[i] == element) {
      size = size - 1;
      for(int j = i; j<size; j++)
        ar[j] = ar[j+1];
    }
    else  i++;
  }
}
```

Рисунок 7.9 – Функція вилучення елементу з масиву

7.2.6 *Функція перевороту масиву*

Текст функції наведено на рисунку 7.10.

```
void transAr(int ar[], int size){
   for(int i=0; i<size/2; i++){
      int tmp = ar[i];
      ar[i]=ar[size-i-1];
      ar[size-i-1]=tmp;
   }
}
```

Рисунок 7.10 – Функція для перевороту масиву

У цій функції елементи масиву, симетрично розташовані відносно середини, міняються місцями. Тому треба перебирати тільки половину масиву.

7.2.7 *Функція формування масиву накопичених значень*

Ця функція створює новий масив, такої ж довжини, як і вихідний, але в цьому масиві кожен елемент дорівнює сумі елементів вихідного масиву від першого до поточного елементу.

Текст функції наведено на рисунку 7.11.

```
void accumAr(int ar[], int ars[], int size){
  ars[0]=ar[0];
  for(int i=1; i<size; i++){
     ars[i]=ars[i-1]+ar[i];
  }
}
```

Рисунок 7.11 – Функція для формування масиву накопичених значень

7.3 ФУНКЦІЇ ОБРОБКИ РЯДКІВ СИМВОЛІВ

Незважаючи на те що за своєю структурою рядки символів майже нічим не відрізняються від масивів чисел, завдання функцій з обробки рядків суттєво відрізняються від завдань функцій з обробки масивів чисел. Так, завдання пошуку суми елементів рядка, або пошук максимального значення елементу рядка не мають ніякого сенсу.

Завдання функцій з обробки рядків зовсім інші, нижче наведено деякі з них:

- копіювання частини рядка;
- вставка рядка у рядок;
- вилучення частини рядка;
- заміна одних символів іншими;
- вилучення деяких символів;
- визначення позиції групи символів у рядку.

7.3.1 *Функція копіювання частини рядка*

```
void subStr(char str[], char sub[], uint index, uint count){
   if (index>strlen(str))count=0;
   if(index+count-1>strlen(str))count=strlen(str)+1-index;
   for(int i=0, j=index; i<count; i++, j++)
      sub[i]=str[j];
   sub[count]='\0';
}
```

7.3.2 *Функція знаходження підрядка у рядку*

```
int posInStr(char str[], char sub[]){
     for(int i=0; i<=strlen(str)-strlen(sub); i++){
       int j=0;
      while(str[i+j]==sub[j] && j<strlen(sub))j++;
      IF (J==STRLEN(SUB))RETURN I;
    }
   return -1;
}
```

7.4 ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

В лабораторній роботі слід створити проект, відповідно до вимог варіантів з таблиць 7.1, 7.2. Номер варіанту вибирайте відповідно до останньої цифри номера залікової книжки.

Таблиця 7.1 передбачає реалізацію таких функцій у проекті:

- створення масиву заданим способом;
- визначення якоїсь числової характеристики для створеного масиву;
- формування нового масиву на основі створеного.

Таблиця 7.1 – Завдання на роботу з масивами

№	Створення масиву	Числові характеристики	Формування нового масиву
1	2	3	4
0	Random	Розмах елементів (max-min)	Видалити парні елементи із масиву
1	Із рядка символів	Різниця між сумами елементів у парних та непарних позиціях	Оборот масиву
2	По елементам	Кількість елементів, що перевищують середнє геометричне	Вставка елемента в задану позицію
3	Random	Різниця між середніми арифметичним та геометричним	Видалення заданого елементу із масиву
4	Із рядка символів	Різниця між сумами парних та непарних елементів	Задане число циклічних зсувів ліворуч
5	По елементам	Середнє відхилення елементів масиву	Вставка суми елементів у початок масиву

Продовження таблиці 7.1

1	2	3	4
6	Random	Середні арифметичні значення парних та непарних елементів	Задане число циклічних зсувів праворуч
7	Із рядка символів	Дисперсія елементів масиву	Вставка середнього арифметичного значення в середину масиву
8	По елементам	Кількість елементів, що перевищують середнє значення	Переформатувати масив, спочатку непарні, потім парні
9	Із рядка символів	Сума елементів, що перевищують середнє значення	Додати мінімальне значення у початок масиву, а максимальне к кінець

Таблиця 7.2 передбачає реалізацію у проекті функції для обробки рядка (масиву символів).

Таблиця 7.4 – Завдання на обробку масивів символів

№	Завдання
0	Отримати задану кількість символів, від початку рядка
1	Отримати задану кількість символів, від кінця рядка
2	Перетворити рядок цифрових символів у число
3	Перетворити ціле число без знаку у прядок символів
4	Вилучити початкові пробіли з рядка символів
5	Вилучити хвостові пробіли з рядка символів
6	Отримати задану кількість символів, починаючи від заданої позиції
7	Видалити задану кількість символів, починаючи від заданої позиції
8	Вставити задану послідовність символів, починаючи від заданої позиції
9	Отримати позицію заданої послідовності символів

7.5 ВИМОГИ ДО ЗВІТУ

- Назва роботи.
- Мета роботи.
- Короткий опис основних понять, пов’язаних масивами.
- Тексти функцій для індивідуальних завдань з коментарями.
- Результати тестування проекту у вигляді копій консолі.
- Висновки.

7.6 КОНТРОЛЬНІ ПИТАННЯ

- Визначення поняття масив.
- Оголошення масиву і його ініціалізація.
- Особливості масивів символів та їх ініціалізації.
- Масиви як параметри функцій.
- Написати функцію для обробки масиву за вказівкою викладача.
- Написать функцію відповідно до одного із варіантів індивідуальних завдань до лабораторної роботи.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программирования С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с
2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. –1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вильямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

8 ЛАБОРАТОРНА РОБОТА № 8. СОРТУВАННЯ МАСИВІВ

Мета роботи:

- Ознайомитися з поняттям сортування масиву.
- Ознайомитися з алгоритмом сортування масиву методом вибору.
- Ознайомитися з алгоритмом сортування масиву методом обміну.
- Ознайомитися з алгоритмом сортування масиву методом вставки.
- Ознайомитися з операціями над впорядкованими масивами.
- Створити програму, яка впорядковує масив та обробляє його відповідно до індивідуального завдання.

8.1 МЕТОДИ СОРТУВАННЯ МАСИВІВ

При роботі з масивами часто виникає необхідність розміщення елементів у зростаючому або спадаючому порядку. Уявіть, наскільки важко було б користуватися словником, якби слова в ньому не розташовувалися в алфавітному порядку. Так само і швидкість та простота алгоритмів обробки масивів багато в чому залежать від порядку, в якому їх елементи зберігаються в пам'яті комп'ютера.

Сортування масиву – це процес перестановки елементів масиву з метою розміщення елементів масиву в певному порядку.

Наприклад, якщо сортується масив A чисел по зростанню, то після сортування цього масиву буде виконуватися умова:

A[0] < A[1] < A[2] < A[3] < …….< A[n]

Найчастіше задачі сортування вирішуються в інформаційних пошукових системах, бо пошук у впорядкованих масивах проводиться набагато швидше, ніж у невпорядкованих.

Існують різні методи сортування масивів. Тут ми розглянемо найпростіші з них, а саме:

- сортування методом вибору.
- сортування методом обміну (метод бульбашки);
- сортування методом вставки.

8.1.1 *Сортування вибором*

Алгоритм сортування елементів масиву у зростаючому порядку за методом вибору можна описати так:

1. Серед усіх елементів масиву, починаючи з першого, шукають мінімальний елемент.
2. Знайдений мінімальний елемент міняють місцями з першим елементом.
3. Переглядають масив від другого елементу, і знаходять мінімальний серед цих елементів.
4. Знайдений елемент міняють місцями з другим елементом.
5. Далі те саме з третім елементом, і так далі до останнього елементу.

Аналіз описаних вище дій показує, що для програмної реалізації цього

методу сортування буде потрібно два цикли for.

У зовнішньому циклі повинен змінюватися номер елементу, куди буде заноситися черговий мінімальний елемент. Номери цих елементів мають змінюватися від першого до передостаннього. Цей цикл буде визначати кількість проходів по масиву.

Внутрішній цикл повинен забезпечити послідовне порівняння елементу, зафіксованого першим циклом, з усіма елементами, які слідують в масиві за ним.

У тілі внутрішнього циклу відбувається порівняння елементів, індекси яких задаються параметрами зовнішнього і внутрішнього циклу. Якщо в результаті порівняння знаходиться елемент, що менший ніж зафіксований, то порівнювані елементи міняються місцями.

Схема алгоритму сортування масиву методом вибору показана на рисунку 8.1.

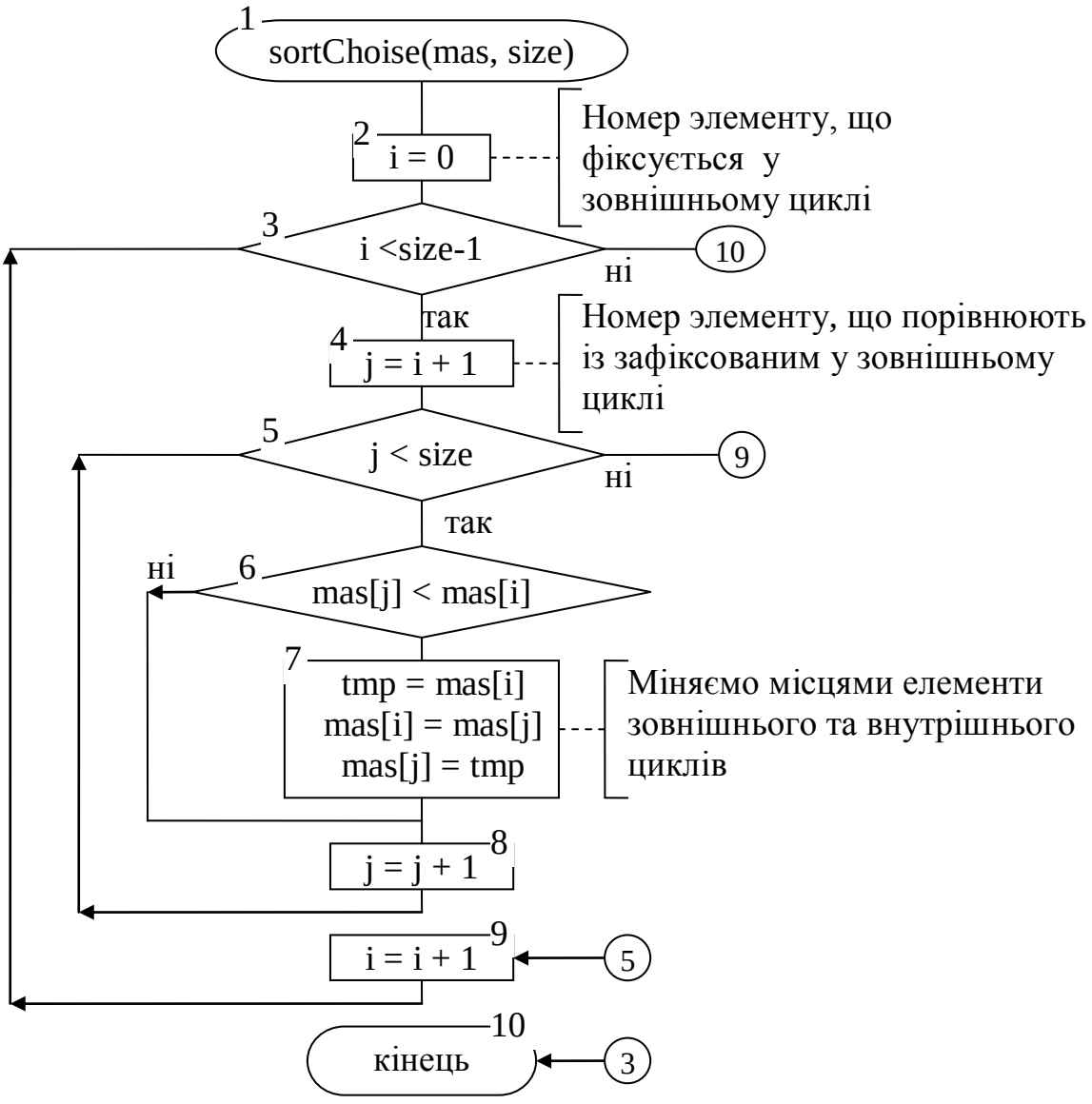


Рисунок 8.1 - Алгоритм сортування елементів масиву у зростаючому порядку за методом вибору

Вихідними даними для алгоритму є: сортований масив mas і кількість елементів у цьому масиві size.

8.1.1.1 Приклад сортування елементів масиву у зростаючому порядку за методом вибору

Починаємо з першого елементу, який назвемо змінюваним, тому що його значення буде змінюватися.

Змінюваний елемент порівнюємо по черзі з кожним з елементів масиву, які йдуть за ним. Ці елементи будемо називати поточними. Якщо значення поточного елементу виявляється менше значення змінюваного, то ці елементи обміняються значеннями.

Після перегляду усіх поточних елементів змінюваний елемент буде мати значення найменше із розглянутих поточних.

Проілюструємо графічно алгоритм цього методу сортування. Початкове розташування елементів масиву показано на рисунку 8.2.

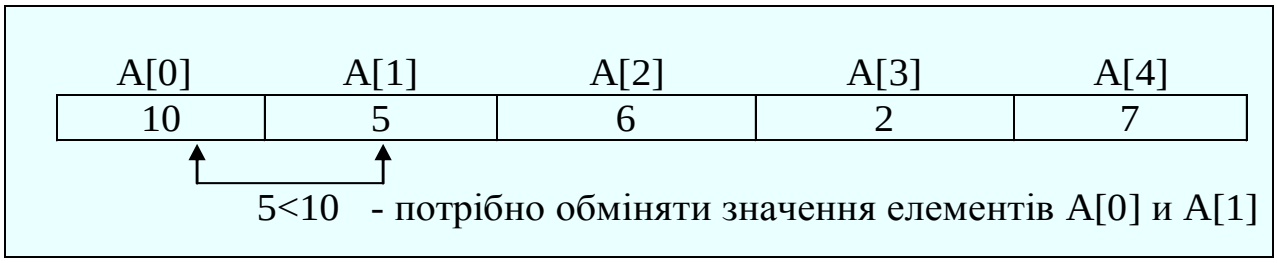


Рисунок 8.2 – Початкова схема масиву, що підлягає сортуванню

Після обміну значеннями елементів A[0] и A[1] масив стане таким, як показано на рисунку 8.3.

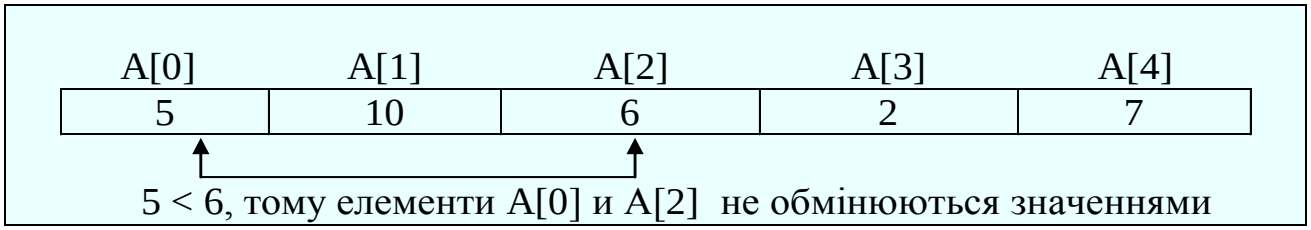


Рисунок 8.3 – Схема масиву після першого обміну значеннями

Далі будуть порівнюватися елементи A[2] и A[0], але A[2] не менше за A[0], тому A[2] залишається без змін.

Аналіз елементу A[3] показує, що він менший за A[0], тому знову потрібен обмін значеннями, але вже між A[3] і A[0]. Результат обміну показано на рисунку 8.4.

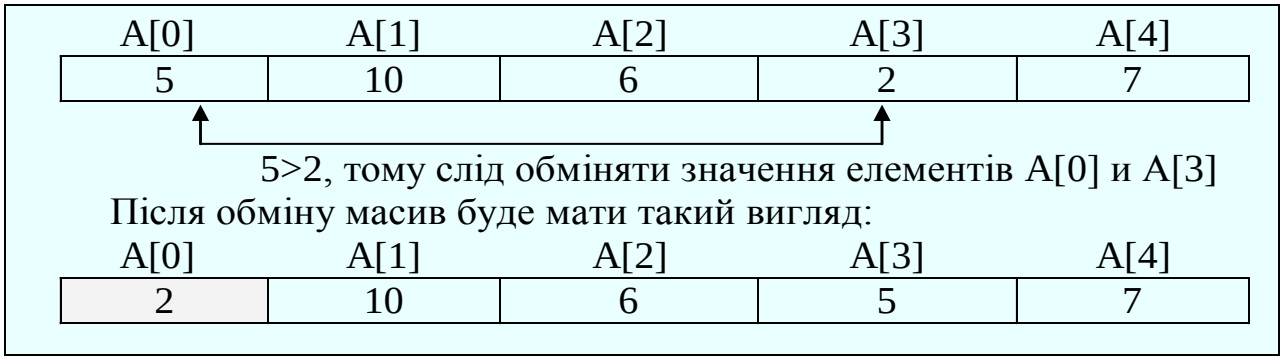


Рисунок 8.4 – Схема масиву після другого обміну значеннями

Далі будуть порівнюватися елементи A [4] і A [0], але A [4] не менший за A[0], тому значення A [5] залишається без змін.

На цьому перший перегляд масиву закінчується. В результаті цього проходу в першій позиції масиву з'явився найменший елемент.

Тепер необхідно здійснити другий перегляд масиву, але в якості змінюваного береться вже другий елемент масиву. З другим елементом будуть порівнюватися елементи, починаючи з третього. В результаті чого на друге місце буде поставлений мінімальний елемент серед елементів з 2-го по 5-й. Результати другого проходу по масиву представлені на рисунку 8.5.

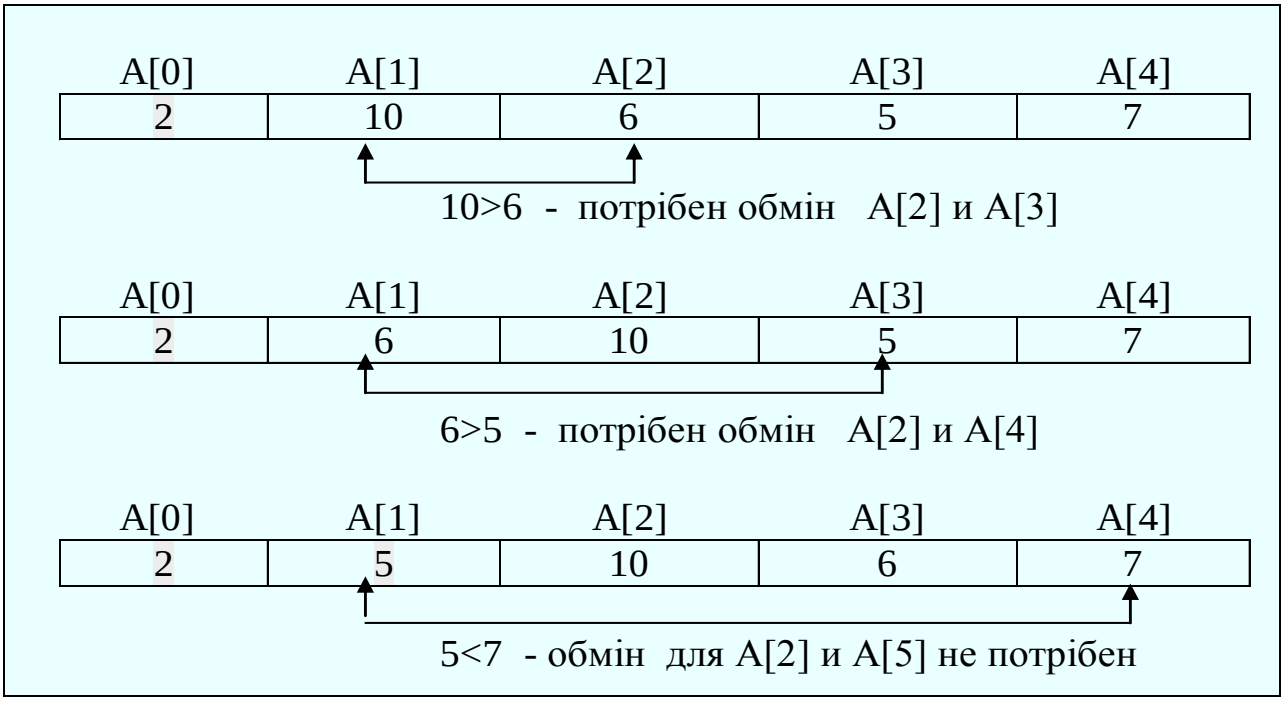


Рисунок 8.5 – Схема другого проходу при сортуванні масиву методом вибору

Під час третього перегляду масиву в ролі змінюваного елементу виступає елемент A[2], і з ним будуть послідовно порівнюватися елементи A[3] і A[4]. Як наслідок, елементи A[2] і A[3] поміняються значеннями.

При останньому, четвертому проході елемент A[3] поміняється значенням з єдиним, наступним за ним, елементом A[4].

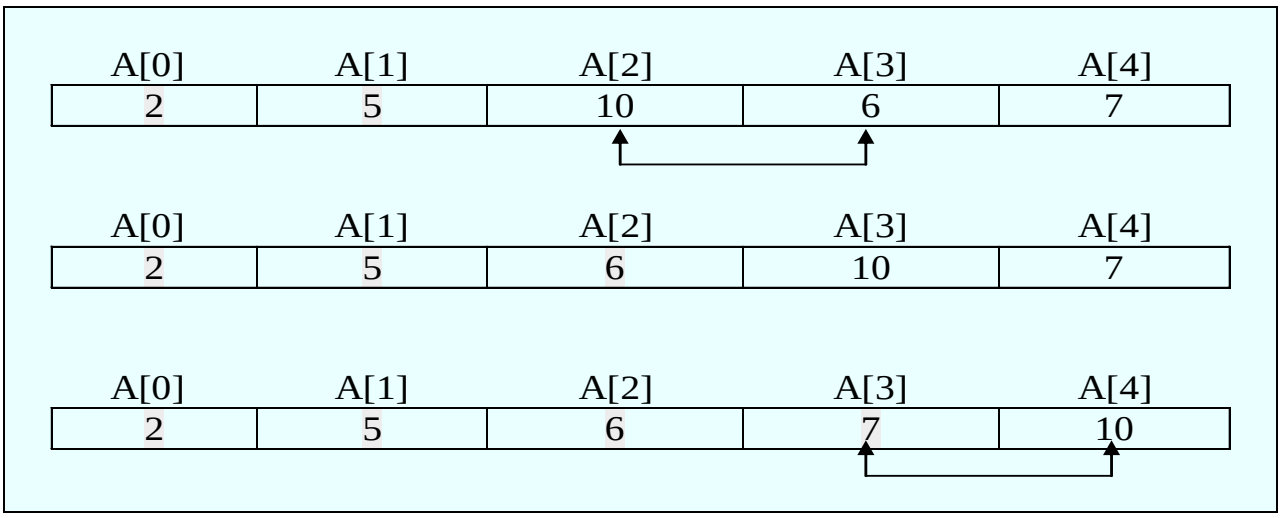


Рисунок 8.6 – Схема останніх проходів по масиву при сортуванні методом вибору

8.1.1.2 Функція сортування масиву методом вибору

Функція, представлена на рисунку 8.7 реалізує за допомогою циклів for алгоритм, зображений на рисунку 8.1.

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

Рисунок 8.7 – Функція сортування елементів масиву у зростаючому порядку за методом вибору

Наведена функція не єдиний варіант реалізації методу вибору.

Для зменшення кількості операцій перезапису можна у внутрішньому циклі тільки запам’ятовувати індекс мінімального елементу. А обмін робити після завершення внутрішнього циклу. Саме такий спосіб слід реалізувати у проекті до лабораторної роботи.

8.1.2 *Сортування обміном (метод бульбашки)*

Алгоритм заснований на принципі порівняння та обміну значеннями (у разі необхідності) між сусідніми елементами. Кожен елемент масиву, починаючи з першого, порівнюється з наступним, і якщо він більше

наступного, то елементи міняються місцями. В результаті після першого проходу, при сортуванні на зростання, найбільший елемент виявиться на останньому місці.

У наступному проході все повторюється заново і найбільший серед решти елементів виявляється на передостанньому місці.

Схему алгоритму сортування елементів масиву у зростаючому порядку за методом обміну показана на рисунку 8.8.

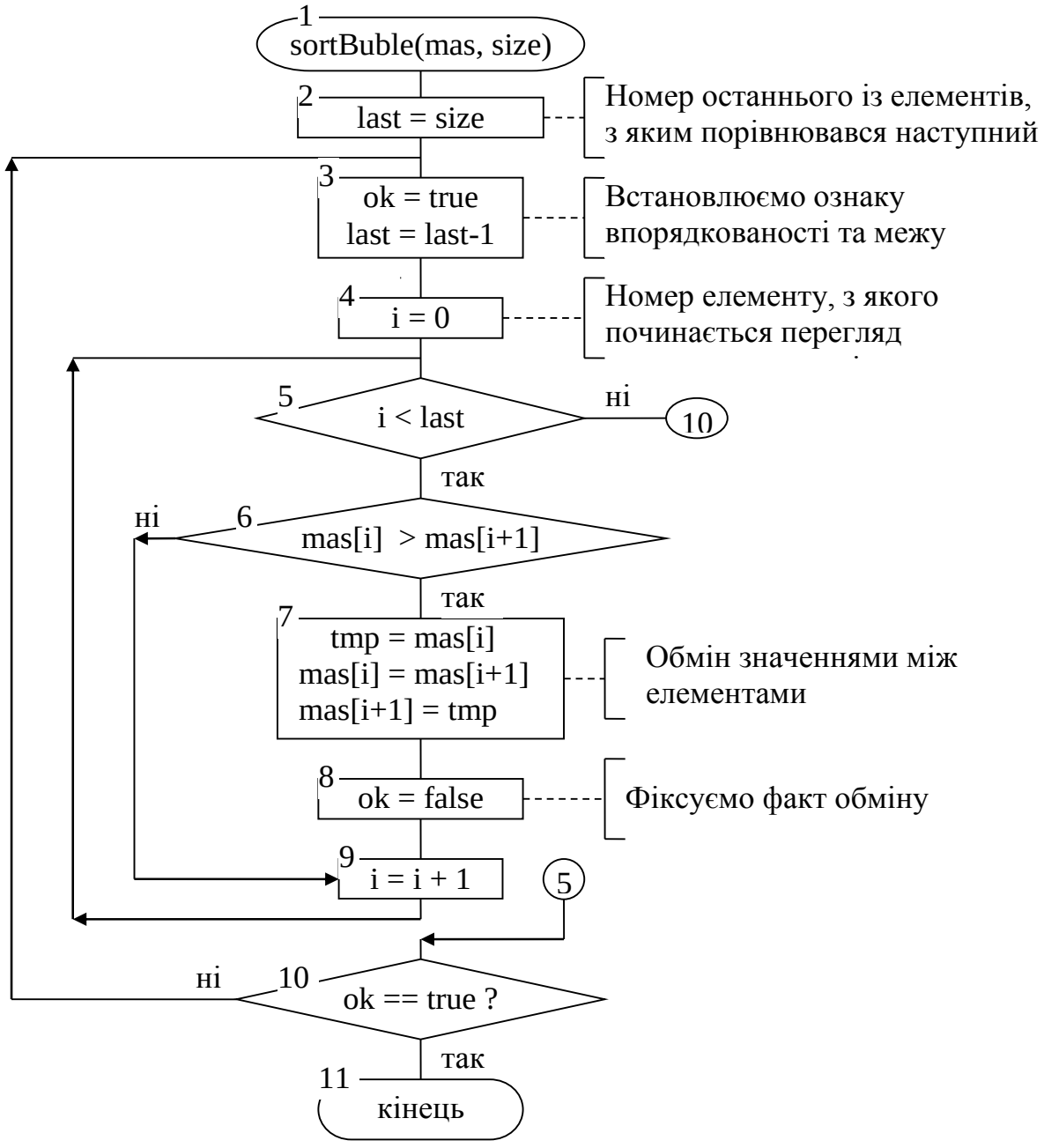


Рисунок 8.8 - Алгоритм сортування елементів масиву у зростаючому порядку за методом обміну

Цей метод отримав ще назву «метод бульбашки». Назва стає зрозумілою, якщо розташовувати масиви вертикально У процесі реалізації алгоритму елементи з меншим значенням просуваються до початку масиву (спливають), а

елементи з великим значенням переміщуються у кінець масиву (тонуть)..

Кожен прохід по масиву від його початку упорядковує щонайменше один елемент у хвості масиву, тому кожного наступного проходу доводиться розглядати менше пар елементів ніж на попередньому проході, бо останні елементи вже впорядковані.

Інколи початкове розташування елементів може бути таким, що масив може бути відсортований за невелику кількість проходів. Ознакою впорядкованості масиву є відсутність обмінів. Тому для скорочення кількості проходів по масиву, слід після закінчення кожного з них перевіряти, чи був зроблений хоч один обмін значень елементів.

У алгоритмі, що зображений на рисунку 8.8, сортування реалізується за допомогою двох циклів. Зовнішній цикл виконується поки по завершенні чергового проходу по масиву не виявиться, що перестановок елементів не було. У цьому випадку змінна ok збереже значення true, і це означатиме, що масив відсортований. Наявність обмінів між елементами при проході по масиву можна перевірити тільки після завершення проходу, тому зрозуміло, що в даному випадку доречним буде цикл do ... while.

У внутрішньому циклі послідовно змінюється індекс елементів, що порівнюються. Тому для організації проходу по масиву використовується цикл for. Максимальне значення параметра цього циклу у кожному проході зменшується на 1.

8.1.2.1 Приклад сортування елементів масиву у зростаючому порядку за методом обміну

На рисунках 8.9 – 8.13 детально показані зміни масиву в процесі сортування обміном під час першого проходу.

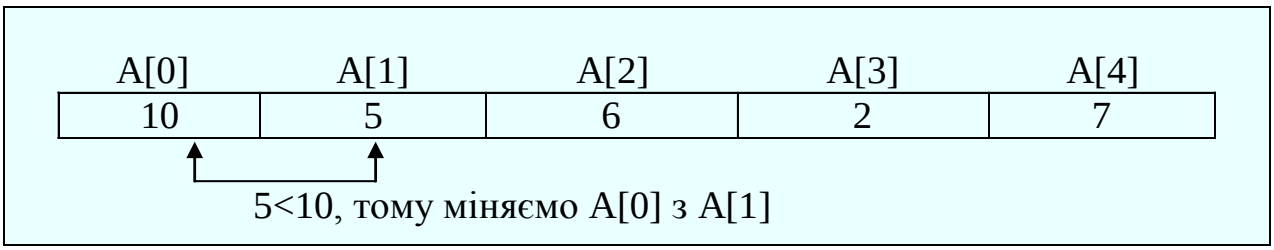


Рисунок 8.9 – Масив до сортування

Рисунок 8.10 – Масив після першого обміну елементів

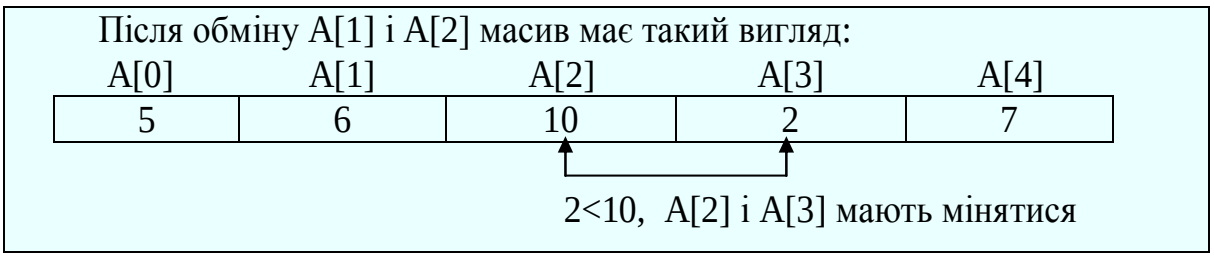


Рисунок 8.11 – Масив після другого обміну елементів

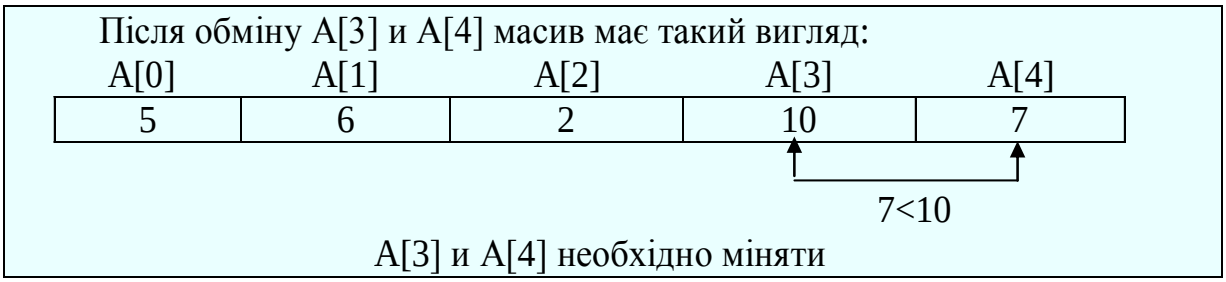


Рисунок 8.12 – Масив після третього обміну елементів

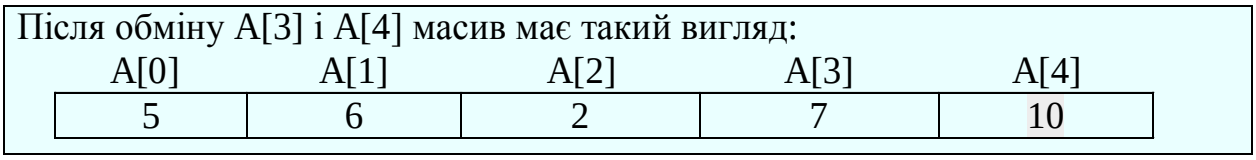


Рисунок 8.13 – Масив після першого проходу

Після першого проходу максимальний елемент став на своє місце у кінці масиву, тому в другому проході по масиву цей елемент уже аналізуватися не буде, тобто число порівнянь в другому проході буде на одне менше.

Другий прохід по масиву розглянемо менш детально. Його результати представлені на рисунку 8.14.

Рисунок 8.14 – Другий прохід по масиву

У другому проході знадобився лише один обмін, і в результаті не тільки A[3], але й A [2] виявилися на своїх місцях.

Результати останнього проходу по масиву показані на рисунку 8.15.

Рисунок 8.15 – Результати третього проходу по масиву

Після третього проходу всі елементи опинилися на своїх місцях. У загальному випадку для сортування масиву знадобився б ще один прохід, але у даному випадку сортування вже закінчено. У цьому особливість сортування обміном – число проходів може бути менше максимально можливого.

8.1.2.2 Функція сортування масиву методом обміну

Функція, представлена на рисунку 8.16 реалізує за допомогою циклів for алгоритм, зображений на рисунку 8.1.

```
void sortBubl(int ar[], int size){
int  last=size; bool ok;
do{
   last=last-1;
   ok = true;
   for(int i= 0; i<last; i++){
      if(ar[i] > ar[i+1]){
         int x = ar[i];
         ar[i] = ar[i+1];
         ar[i+1] =  x;
         ok =  false;
      }
   }
}while(!ok);
}
```

Рисунок 8.16 - Функція сортування елементів масиву у зростаючому порядку за методом обміну

Роботу функції, представленої на рисунку 8.16 можна прискорити, якщо після обміну елементів фіксувати номер елементу, що останнім змінювався (значення змінної і). Це значення можна використовувати і як ознаку завершення сортування, так і в якості максимального значення параметру внутрішнього циклу. Це удосконалення алгоритму дозволяє дещо зменшити кількість циклів сортування. Саме такий спосіб слід реалізувати у проекті до лабораторної роботи.

8.1.3 *Сортування вставкою*

Суть алгоритму полягає у наступному. На кожному кроці елементи масиву поділені на впорядковану частину A[1], A[2], ..., A[i-1], яка розташовується на початку масиву, та невпорядковану частину A [i], ... , a [N], і перший елемент з невпорядкованої частини вставляється в упорядковану. На початку сортування впорядкована частина складається всього з одного, першого елемента, а всі інші елементи розташовуються в другій частині масиву. Схема цього алгоритму наведена на рисунку 8.17,

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 вилучається і переноситься в упорядковану частину. Місце для цього елементу звільняється шляхом зсуву упорядкованих елементів вправо на місце витягнутого елемента. Зсув упорядкованих елементів на одну позицію вправо триває, поки не буде знайдено місце для елемента, вилученого з невпорядкованої послідовності.

Як видно з рисунку 8.17, в алгоритмі є 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 забезпечує послідовні зсуви упорядкованих елементів вправо, починаючи з останнього, доти, поки не буде знайдено місце для першого елемента з невпорядкованою області, що зберігався у змінній tmp.

Можливість вставки елементу визначається однією з двох умов.

- (mas[j-1] <= buf < mas[j]),якщо (1 < j < i), тобто знайдено місце всередині впорядкованої послідовності.
- j=1 , тобто. tmp менше ніж усі елементи впорядкованої частини і має бути поставлений на перше місце у впорядкованій частині масиву.

Після виконання однієї з цих умов цикл зсувів завершується і елемент масиву із змінної tmp переноситься на знайдене місце у впорядкованій послідовності.

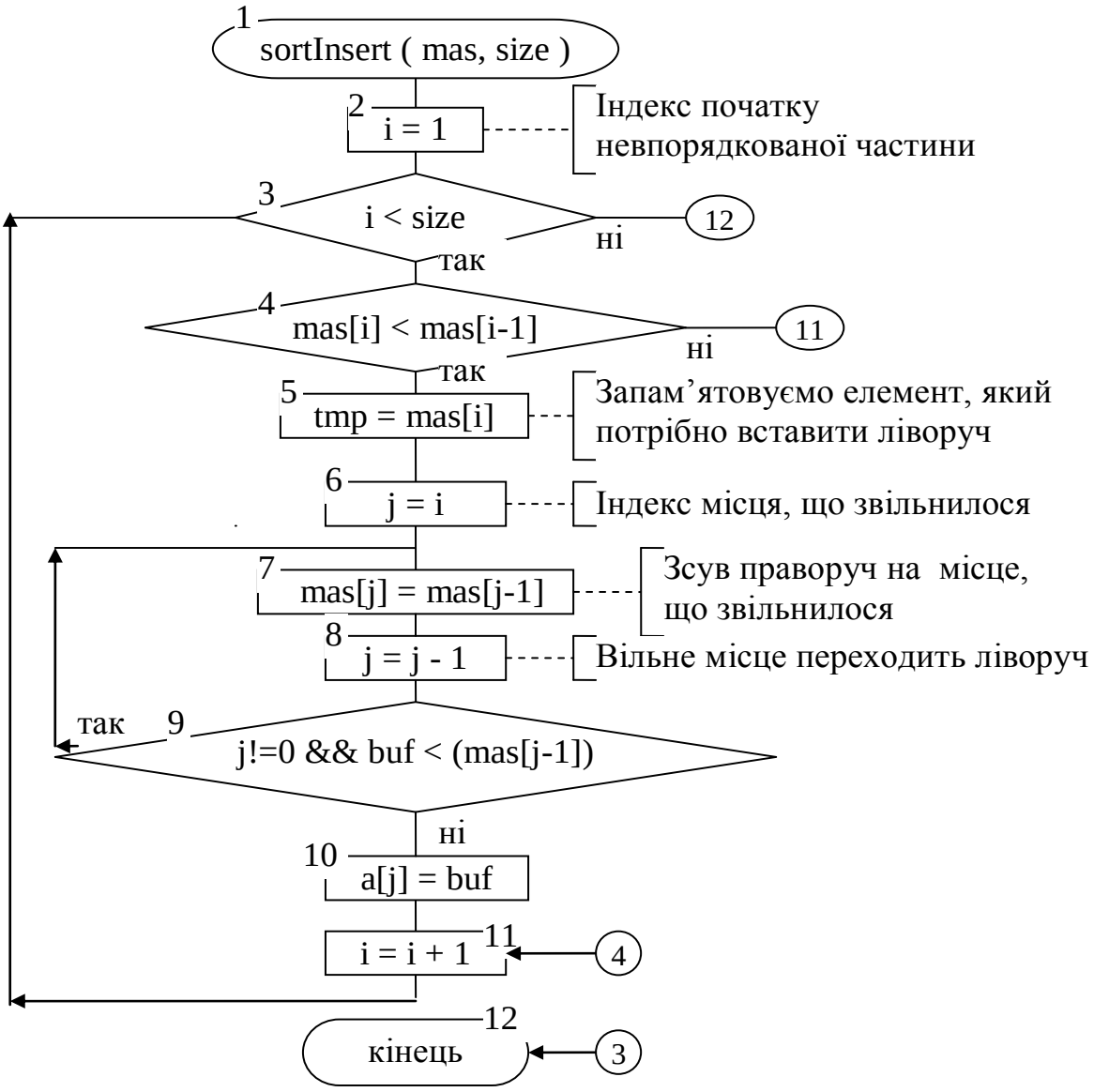


Рисунок 8.17 - Алгоритм сортування елементів масиву у зростаючому порядку за методом вставки

8.1.3.1 Приклад сортування елементів масиву у зростаючому порядку за методом вставки

Результати послідовних кроків сортування представлені на рисунках 8.18 - 8.22.

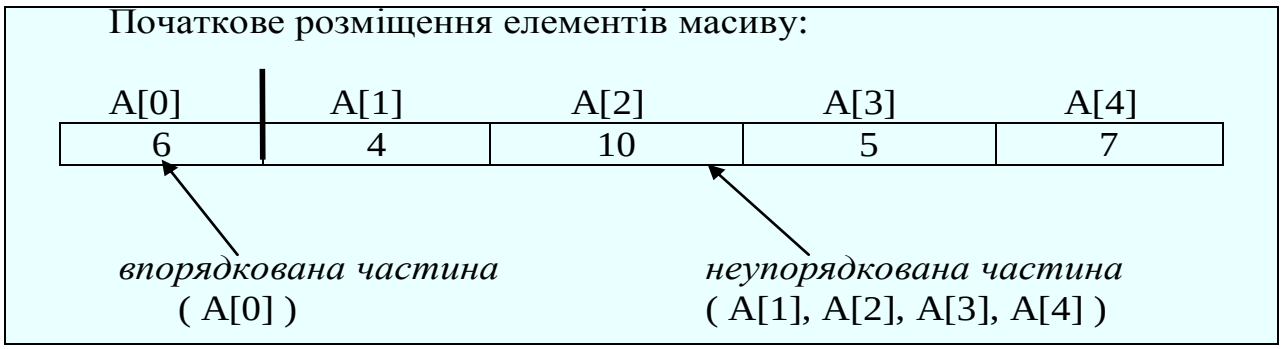


Рисунок 8.18 – Початкове розбиття масиву при сортуванні за методом вставки

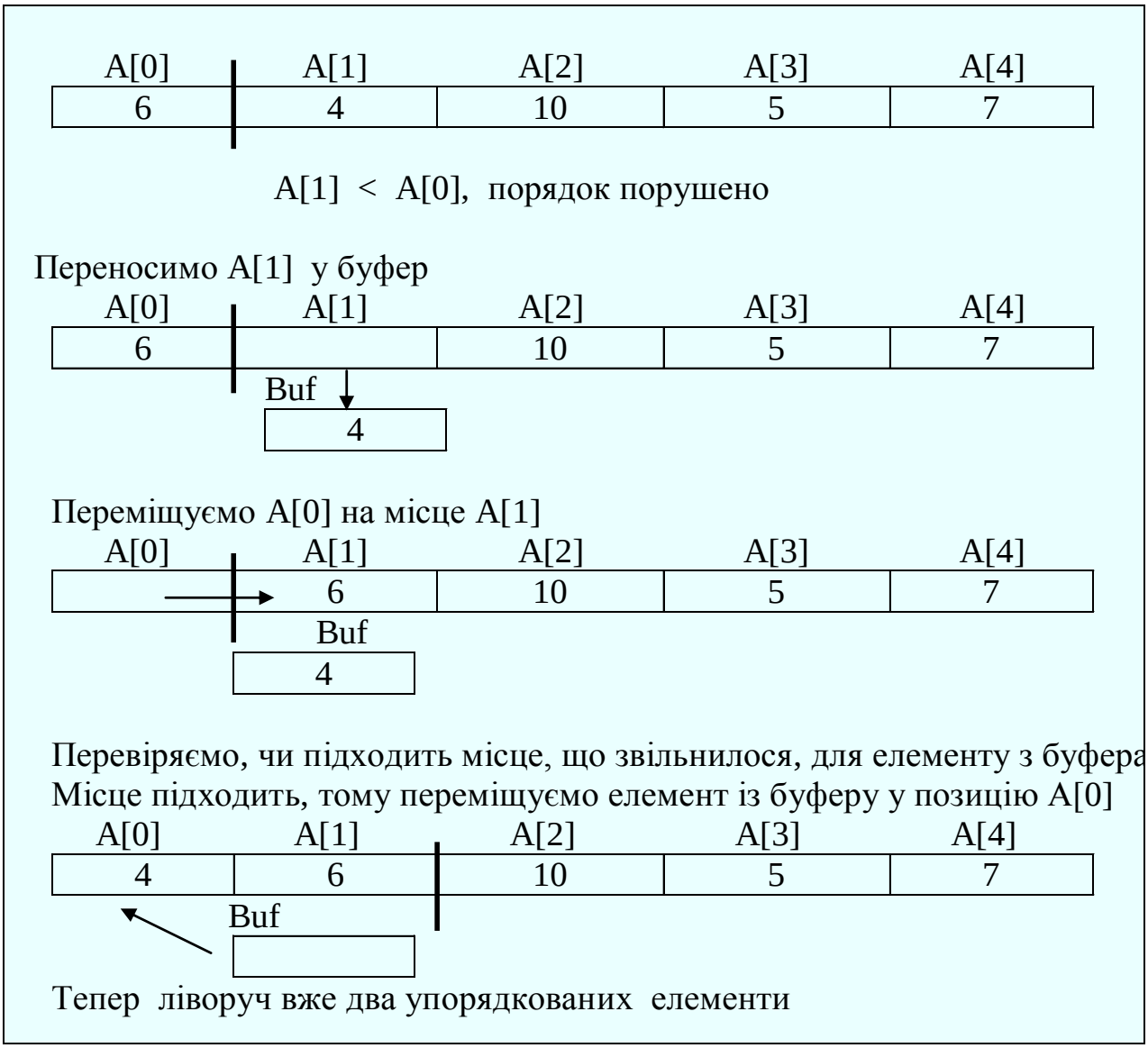


Рисунок 8.19 – Результати виконання першого шагу сортування вставкою

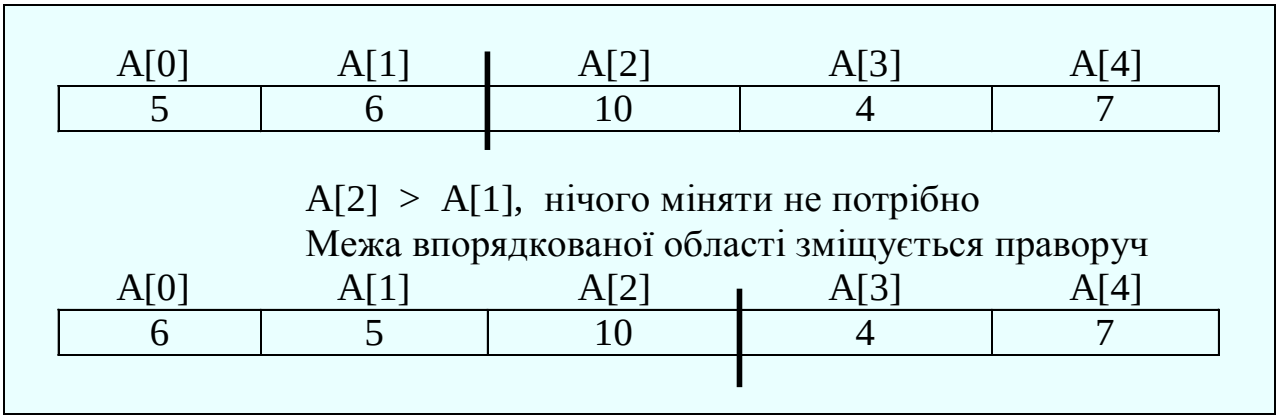
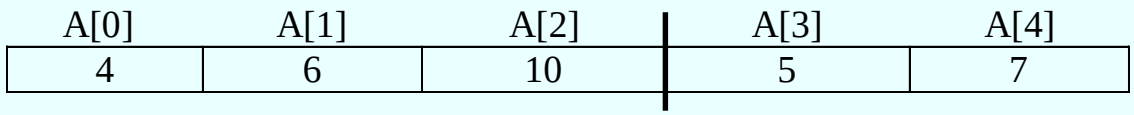
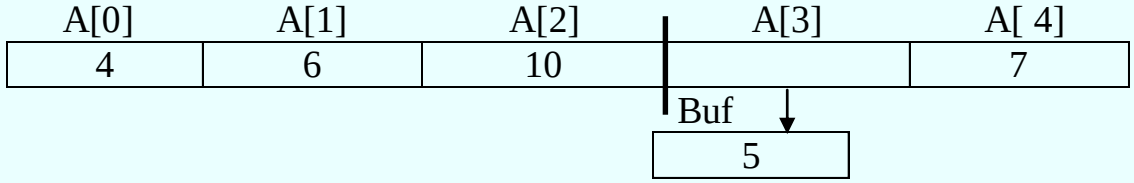


Рисунок 8.20 – Результати виконання другого шагу сортування вставкою

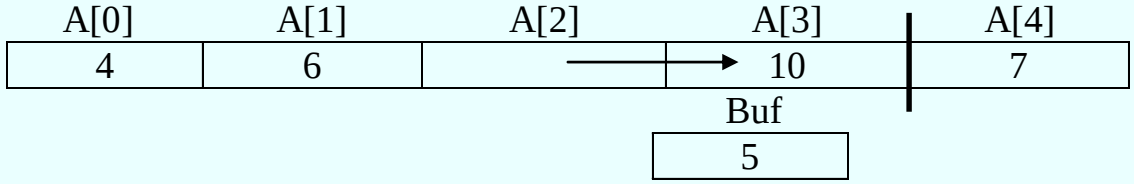


A[3] < A[2], порядок порушено

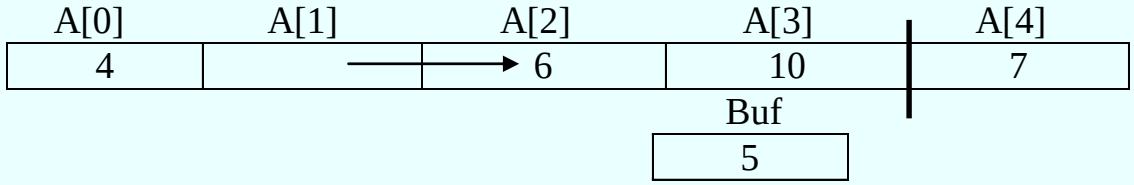
Переносимо A[3] у буфер



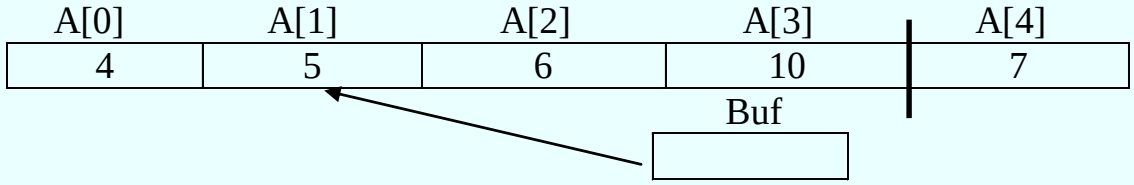
Переміщуємо A[2] на место A[3]



Перевіряємо, чи підходить місце, що звільнилося, для елементу з буфера.
Місце не підходить, тому переміщуємо A[1] у позицію A[2].



Перевіряємо, чи підходить місце, що звільнилося, для елементу з буфера.
Місце підходить, тому переміщуємо із буфера у позицію A[1].



Тепер ліворуч вже чотири елементи, що впорядковані.

Рисунок 8.21 – Результати виконання третього шагу сортування вставкою

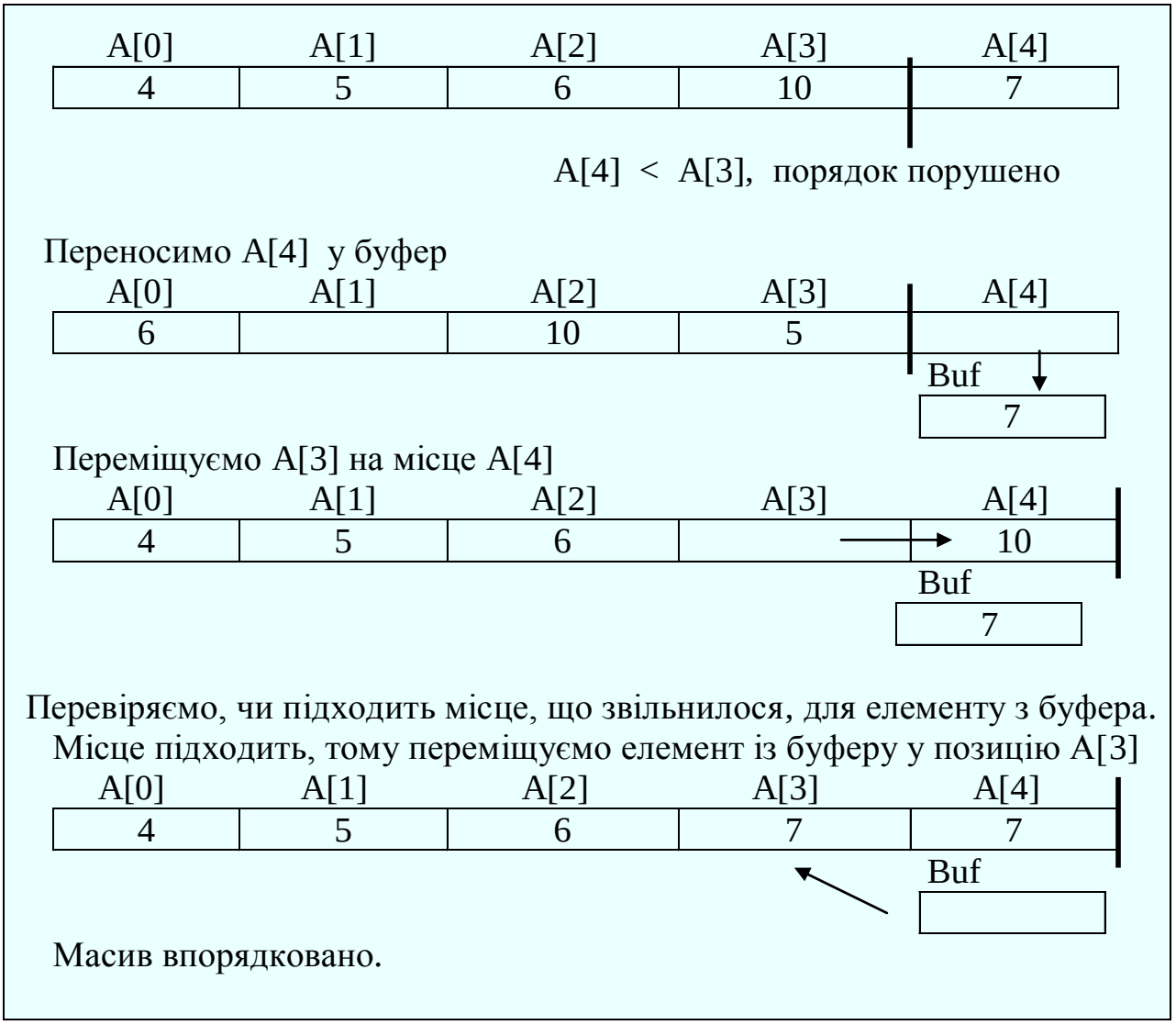


Рисунок 8.22 – Результати виконання четвертого шагу сортування вставкою

8.1.3.2 Функція сортування масиву за методом вставки

Програмна реалізація алгоритму за методом вставки наведена на рисунку 8.23.

Ефективність роботи цієї функції можна дещо підвищити за рахунок зменшення кількості порівнянь під час пошуку місця для елементу із буфера. Зважаючи на те, що ліва частина масиву вже впорядкована, місце вставки для елементу із буфера можна знайти методом дихотомії. Цей метод буде розглянуто у підрозділі 8.3. Суть методу полягає в тому, що елемент, який знаходиться в середині області пошуку, порівнюється із елементом у буфері. В результаті порівняння можна визначити, в якій з половин області пошуку знаходиться місце для елементу із буфера, праворуч або ліворуч. Таким чином, в результаті одного порівняння область пошуку звужується наполовину.

Місце вставки буде знаходитися за останнім значенням правої межі області пошуку.

Визначивши місце вставки, можна робити зсув елементів лівої частини масиву праворуч і вставляти елемент із буфера.

Саме таку реалізацію алгоритму слід зробити у проекті до лабораторної роботи.

```
void sortInsert(int ar[], int size){
   for(int i=1; i<size; i++){
      //Порівнюємо елементи на межі між частинами масиву
      if(ar[i] < ar[i-1] ){
         int buf = ar[i];   // Порядок порушено, переносимо і-й елемент у буфері
         // Починаємо зсуви праворуч на місце і-го елементу
         int j = i;   // j – індекс вільного місця
         do{
               ar[j]=ar[j-1];   // зсув праворуч
               j=j-1;
               // поки ліворуч число більше,  або дійшли до початку масиву
           }while (j != 0 && buf <ar[j-1] );
         //Вставляємо елемент, що мав номер і на нове місце з индексом j
         ar[j] = buf;
      }
   }
}
```

Рисунок 8.23 – Функція сортування елементів масиву у зростаючому порядку за методом вставки

8.2 СОРТУВАННЯ ЗА УСКЛАДНЕНИМИ ПРАВИЛАМИ

Вище ми розглянули різні методи сортування масивів, в яких порядок розташування елементів визначався застосуванням найпростішої операції порівняння «більше» або «менше». В результаті застосування цих операцій ми отримуємо сортування на зростання або на спадання.

Однак у деяких випадках необхідно використовувати більш складні критерії порівняння елементів, ніж просте порівняння. У цих випадках доцільно написати функцію порівняння елементів по заданому правилу, яка буде повертати true, якщо порядок проходження елементів не порушений і false у противному випадку. Цю функцію слід викликати в тих місцях процедури сортування, де потрібно порівняння елементів.

Наприклад, нехай правило сортування масиву цілих чисел

сформульовано таким чином: «Спочатку парні числа за спаданням, а потім непарні за зростанням».

Функцію, яка порівнює елементи таким чином, наведено нижче:.

```
bool goodDisposition(int x1, int x2){
   if (x1 % 2 != x2 % 2)
        //Якщо одне число парне а друге непарне, то меншим буде парне
        return x1 % 2 < x2 % 2;
   else if( x1 % 2 == 0)  // Числа парні, має бути спадання
            return x1>x2;
      else  // Числа непарні, має бути зростання
            return x1 <  x2;
}
```

Тепер за допомогою створеної функції можна відсортувати масив в потрібному порядку. Метод сортування може бути будь-яким. Для прикладу використаємо сортування вибором:

```
void sortChoise(int ar[], int size){
   for(int i=0; i<size-1;i++){
      for(int j = i + 1; j<size; j++) {
         if (!goodDisposition(ar[i] ,ar[j])){
            int x =  ar[i];
            ar[i] =  ar[j];
            ar[j] =  x;
         }
      }
   }
}
```

8.3 ОБРОБКА УПОРЯДКОВАНИХ МАСИВІВ

Впорядковані масиви найчастіше використовуються як сховища деякої інформації. Найчастіше зустрічаються такі завдання, пов'язані з їх обробкою:

- пошук позиції елемента в масиві;
- вставка елементу у масив, без порушення порядку;
- видалення елемента з масиву;
- об'єднання двох масивів в один зі збереженням порядку;

Нижче розглядаються функції, які вирішують ці завдання.

8.3.1 *Пошук позиції елемента у впорядкованому масиві*

Для пошуку позиції елемента у впорядкованому масиві можна використовувати метод дихотомії (ділення області пошуку навпіл). У цьому методі елемент, який знаходиться в середині області пошуку, порівнюється із зразком, який потрібно знайти. Якщо він не відповідає зразку, то по його значенню можна визначити, в якій з половин області пошуку може знаходитися

потрібний елемент, праворуч або ліворуч. Таким чином, в результаті одного порівняння область пошуку звужується наполовину.

Цикл пошуку повторюється доти, поки потрібний елемент не буде знайдений, або область пошуку звузиться до одного елемента, що буде свідчити про те, що потрібного елементу в масиві немає.

Функція, яка повертає номер позиції елементу у впорядкованому масиві наведена на рисунку 8.24. Якщо елемент не знайдено, функція повертає -1.

```
int findPos(int element, int ar[], int size){
   // left, right – лівая та права межі області пошуку
   int left = 0, right = size-1;
   while (left <= right){
      // Обчислюємо індекс середини області пошуку
      int pos=(right + left)/2;
      if(ar[pos] == element)
         return pos; //елемент знайдено
      if  (element < ar[pos])
         right = pos-1; // елемент знаходиться ліворуч
      else left = pos + 1; //елемент знаходиться праворуч
   }
   // Елемент не знайдено
   return -1;
}
```

Рисунок 8.24 – Функція пошуку позиції елемента у впорядкованому масиві

8.3.2 *Вставка елементу до впорядкованого масиву*

Алгоритм вставки елементу до впорядкованого масиву вже розглядався, як частина алгоритму сортування вставкою. Він полягає у послідовному аналізі елементів масиву, починаючи з останнього. Якщо елемент масиву більший ніж той, що потрібно вставити, елементи масиву переміщують вправо на одну позицію, для того, щоб звільнити місце для елемента, що вставляється. Зсуви проводяться доти, поки не буде знайдено місце, відповідне значенню елемента, що вставляється.

Якщо всі елементи масиву більше ніж елемент, що додається, то всі елементи масиву перемістяться праворуч, а новий буде поставлений на перше місце. Зрозуміло, що кількість елементів масиву при цьому збільшується на один.

Функція вставки елемента до впорядкованого масиву таким методом наведена на рисунку 8.25.

```
void addToSortAr(int element, int ar[], int &size){
   int i = size - 1;
   while( ar[i]>element &&  i>=0 ){
      ar[i+1] = ar[i];
      i--;
   }
   ar[i+1] = element;
   size += 1;
}
```

Рисунок 8.25 – Функція вставки елементу до впорядкованого масиву

Але функцію, наведену на рисунку 8.25 можна дещо прискорити за рахунок зменшення кількості порівнянь, використавши метод дихотомії для пошуку місця вставки.

Такий варіант функції вставки наведено на рисунку 8.26. Як бачимо, спроба підвищити ефективність алгоритму приводить до його ускладнення.

```
void addToSortArray(int element, int ar[], int &size){
   // left і right лівая і права границі області пошуку
   int left = 0, right = size-1;
   while (left <= right){
      // Обчислюємо індекс середини області пошуку
      int pos=(right + left)/2;
      if  (element < ar[pos] )
         right = pos-1;// потрібна позиція заходиться ліворуч
      else left = pos + 1;//потрібна позиція заходиться праворуч
   }
   //Позиція вставки знаходиться після елементу з індексом right
   int i;
   for( i=size; i>right; i--)
        ar[i+1]=ar[i];
   ar[i+1]=element;
   size+=1;
}
```

Рисунок 8.26 – Функція вставки елементу до впорядкованого масиву, яка використовує метод дихотомії

8.3.3 *Видалення елементу з упорядкованого масиву*

В алгоритмі видалення елементу з упорядкованого масиву можна використовувати розглянуту вище функцію для пошуку індексу елементу, що видаляється. Після визначення цього індексу, елемент видаляється шляхом зсуву ліворуч на одну позицію всіх елементів, що знаходяться за тим, який потрібно видалити. Значення змінної, в якій зберігається кількість елементів, у кожному циклі зменшується на одиницю.

Процедура видалення елементу з упорядкованого масиву наведена на рисунку 8.27. Процедурою передбачено, що у масиві можуть бути однакові елементи.

```
bool delFromSortAr(int element, int ar[], int &size){
   int pos;
   while(( pos = findPos(element, ar,size))!= -1) {
      size--;
      for(int i=pos;i<size;i++)
         ar[i]=ar[i+1];
   }
   return pos!=-1 ? true : false;
}
```

Рисунок 8.27 – Функція видалення елементу із впорядкованого масиву

8.3.4 *Злиття двох впорядкованих масивів*

Функція злиття масивів наведена на рисунку 8.28.

У алгоритмі, що реалізує наведена функція, поточні елементи вихідних масивів порівнюються, і у новий масив переноситься менший елемент. При цьому поточна позиція масиву, з якого був переписаний елемент, переміщується до наступного елементу. Цикл порівнянь продовжується поки не закінчаться елементи одного з масивів. Після цього елементи, що залишилася у другому масиві просто дописуються у новий масив.

Зверніть увагу на те, як у функції використовуються операція постфіксного інкременту в операціях присвоєння. Для присвоєння використовуються старі значення індексів, а після присвоєння значення індексів збільшуються на 1.

Після того, як один із масивів вичерпався, дописування решти з другого масиву здійснюється за допомогою циклу for, у якому одночасно змінюються обидва індекси і в якості перших індексів використовуються ті самі змінні і їх останні значення у попередньому циклі.

```
void joinSortArrays(int ar1[], int ar2[], int ar3[], int size1, int size2, int &size3){
   size3 = size1 + size2;
   int k=0, i=0, j=0; // поточні індекс для першого, другого та третього масивів
   while (i < size1 && j < size2) {
      if (ar1[i] < ar2[j])  // Добавляємо елемент із першого масиву
         ar3[k++] = ar1[i++];
      else  // Добавляємо елемент із другого масиву
         ar3[k++] = ar2[j++];
    }
   for(i= i;i< size1; i++, k++)  // Додаємо залишок з першого массиву, якщо він є
      ar3[k] = ar1[i];
   for(j = j; j<size2; j++, k++) // Додаємо залишок з другого массиву, якщо він є
      ar3[k] = ar2[j];
}
```

Рисунок 8.28 – Функція видалення елементу із впорядкованого масиву

8.4 ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

В лабораторній роботі слід створити програму, відповідно до вимог варіанту з таблиці 8.1. Номер варіанту вибирається відповідно до останньої цифри номеру залікової книжки.

Таблиця 8.1 – Завдання для самостійної роботи

№	Початковий масив	Дані масиву	Перший метод сортування	Ускладнене правило сортування
1	2	3	4	5
0	Із рядка символів	±int	Модифікований вибором	Спочатку додатні на збільшення, потім від’ємні на збільшення
1	Random	±int	Модифікований обміном	Спочатку від’ємні на зменшення, потім додатні на зменшення
2	По елементам	±int	Модифікований вставкою	Спочатку парні, потім непарні, усі на зростання
3	Із рядка символів	±int	Модифікований вибором	Спочатку не парні, потім парні, усі на спадання
4	Random	+int	Модифікований обміном	Спочатку непарні за зростанням, потім парні числа за спаданням
5	По елементам	±int	Модифікований вставкою	Спочатку додатні на збільшення, потім від’ємні на збільшення
6	Із рядка символів	±int	Модифікований вибором	Спочатку від’ємні на зменшення, потім додатні на зменшення

Продовження таблиці 8.1

1	2	3	4	5
7	Random	±int	Модифікований обміном	Спочатку парні, потім непарні, усі на зростання
8	По елементам	±int	Модифікований вставкою	Спочатку не парні, потім парні, усі на спадання
9	Random	+int	Модифікований вибором	Спочатку непарні за зростанням, потім парні числа за спаданням

У програмі мають бути реалізовані такі функції:

– створення масиву чисел із заданими параметрами;
– сортування масиву заданим модифікованим методом ;
– сортування за ускладненим правилом;
– вставка елементу до масиву, який упорядковано за ускладненим правилом;
– видалення елементу із такого ж масиву.

Кількість елементів у масиві та параметри масиву має задавати користувач.

Функції сортування у проекті мають бути створені з урахуванням модифікацій, про які згадувалося у теоретичних відомостях.

Студенти, для яких реалізація сортування за ускладненим правилом виявиться складною, можуть сортувати масив за звичайним правилом на зростання або спадання, але в цьому випадку оцінка за проект буде нижче.

8.5 ВИМОГИ ДО ЗВІТУ

– Назва роботи.
– Мета роботи.
– Короткий опис основних понять, пов’язаних з сортуванням та обробкою впорядкованих масивів.
– Тексти функцій для індивідуальних завдань з коментарями.
– Результати тестування проекту у вигляді копій консолі.
– Висновки.

8.6 КОНТРОЛЬНІ ПИТАННЯ

– Сортування за методом вибору.
– Сортування за методом обміну.
– Сортування за методом вставкою.
– Сортування за ускладненими правилами.
– Пошук елементу у впорядкованому масиві.
– Вставка елементу до впорядкованого масиву.
– Видалення елементу з впорядкованого масиву.
– Об’єднання впорядкованих масивів.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программирования С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с
2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. – 1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вильямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

9 ЛАБОРАТОРНА РОБОТА № 9. ДВОВИМІРНІ МАСИВИ (МАТРИЦІ)

Мета роботи:
- Навчитися оголошувати та ініціалізувати двовимірні масиви.
- Опанувати алгоритми тотальної та вибіркової обробки елементів матриць.
- Опанувати алгоритми перетворень матриць.
- Опанувати алгоритми сортування матриць.

9.1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Матриця - це структура даних, якій відповідає таблиця. При роботі з матрицями оперують поняттями ім'я матриці, стовпець, номер стовпця, рядок, номер рядка, елемент, розмір матриці, наприклад 3 х 4.

Кожен елемент матриці має своє значення, але тип усіх елементів матриці однаковий. Для ідентифікації елементів матриці використовують індекси. З поняттям індексу ми вже зустрічалися в попередній роботі, де за допомогою індексу ідентифікувалися елементи одновимірного масиву.

У матриці, для ідентифікації елементу, потрібно два індекси. Це нібито координати елементу в матриці. При зверненні до елементу його індекси вказують після імені матриці в квадратних дужках, розділяючи їх комою. Наприклад, якщо ім'я матриці matrix, то matrix [2,3] - це елемент матриці з координатами 2 і 3. Перший індекс задає номер рядка, а другий - номер стовпця.

Нумерація рядків і стовпців починається з 0.

9.1.1 *Оголошення та ініціалізація матриць*

Оголошення матриці має вигляд, представлений на рисунку 9.1.

<тип елементів> <ім’я матриці> [<кількість рядків>] [<кількість стовпців>];

Рисунок 9.1 – Синтаксис оголошення матриці

Тип елементів визначає тип елементів, з яких складається матриця. Це може бути будь який допустимий у мові тип, простий або складений

Ім’я матриці – це ідентифікатор написаний за правилами запису імен у мові С С++

Кількість рядків та кількість стовпців – це константи, або константні вирази, що визначають розмір матриці.

Приклад оголошення матриці з ім'ям А, що складається з десяти рядків по шість елементів цілого типу у кожномурядку наведено нижче:

```
int A [10] [6];
```

Як і у випадках із одновимірними масивами, під час оголошення матриць можна ініціалізувати усі її елементи або тільки декілька початкових. Приклад оголошення матриці з неповною ініціалізацією тільки трьох рядків із десяти наведено нижче:

```
int  A [10][6] = {{2,5,10}, {6,12}, {7}};
```

Цей приклад показує, що фактично матриця являє собою масив, кожен елемент якого теж є масивом.

Незалежно від того, скільки елементів матриці було ініціалізовано при оголошенні, пам’ять виділяється під усі елементи. Значення елементів, що не були ініціалізовані, невизначені («сміття») або дорівнюють нулю, якщо матриця визначена як глобальна чи статично.

Якщо у оголошенні матриці ініціалізуються усі елементи (повна ініціалізація), то елементи матриці можна записувати не поділяючи їх на рядки. Приклад такого оголошення наведено нижче:

```
int  A [3][4] ={2, 5, 10, 3, 6, 0, 9, 4, 5, 7, 12, 1};
```

У оголошенні матриці при повній ініціалізації кількість рядків можна не показувати, бо їх кількість може бути обчислена виходячи із загальної кількості елементів та розміру рядка. Приклад такого оголошення наведено нижче:

```
int  A [][4] ={2, 5, 10, 3, 6, 0, 9, 4, 5, 7, 12, 1};
```

Як уже було сказано, для доступу до елементів матриці використовується синтаксична конструкція, що складається з імені матриці та пари індексів, кожен з яких записується у своїх квадратних дужках. Тобто доступ до елементів цього матриці з наведеного вище прикладу забезпечується виразом: A[i][j]. Індекс і в даному прикладі є цілим числом у діапазоні від 0 до 2, а індекс j – це ціле число від 0 до3. Таким чином, A [0][0] - це ім'я першого елемента, а A [2][3] - ім'я останнього елементу.

Індексовані елементи матриці можуть бути використані так само, як і прості змінні. Наприклад, вони можуть перебувати у виразах як операнди, їм можна привласнювати будь-які значення, відповідні їх типу.

Як і у випадках із одновимірними масивами слід пам’ятати, що ніякого контролю за значеннями індексів, що використовуються для доступу до елементів матриці нема. Ви можете звернутися до «елементу матриці» з індексами, значення яких перевищують граничні, але отримаєте невідомо що. Ще гірше, якщо ви зміните значення цього «елементу масиву». Це може призвести до катастрофічних наслідків для вашої програми.

9.1.2 *Матриці як параметри функцій*

Якщо формальним параметром функції є матриця, то він оголошується майже так само, як і одновимірний масив. Різниця полягає у тому, що обов’язково потрібно вказувати кількість елементів для другого виміру, тобто довжину рядка матриці.

Кількість елементів у рядку, що наведена у заголовку функції обов'язково має бути тотожною кількості елементів у рядку для матриць, що оголошені у програмі, яка викликає цю функцію, і які передаються у функцію як параметри. Тому, для того щоб уникнути помилок, доцільно у таких програмах оголошувати глобальні константи, що визначають розмір матриці, у вигляді макросів за допомогою директиви #define.

Слід також пам'ятати, що хоча перед іменем матриці, як формального параметру, символ & не ставиться, та все одно матрицю буде передано за посиланням. А символ & ставити не потрібно тому, що ім'я матриці і є адресою першого її елементу.

Крім того, слід брати до уваги той факт, що матриця «не знає», скільки рядків у неї реально заповнено, і скільки елементів у рядку, тому передаючи масив до функції слід передавати і кількість рядків і кількість елементів у рядку, що мають бути оброблені. Ці числа, звичайно, не можуть бути більшими ніж значення під які було виділено пам'ять під час оголошення матриці.

Якщо матриця є результатом, який має повернути функція, то такий результат, як і у випадках із масивами ми поки що будемо повертати через параметри функції.

Приклади функцій, що працюють з матрицями розглянуто у наступних пунктах.

9.1.3 *Формування та виведення матриць з використанням консолі*

Для виведення матриці на консоль можна використовувати функцію, що зображена на рисунку 9.2. Перед описом функції наведені директиви #define, що визначають максимально можливий розмір матриці.

```
#define ROWS 10
#define COLS 10
...
void printMatrix(int matr[][COLS], int nRow, int nCol) {
   cout << endl;
   for(int i = 0; i < nRow; i++) {
      for(int j = 0; j < nCol; j++) {
         cout << setw(6) << matr[i][j];
      }
      cout << endl;
   }
}
```

Рисунок 9.2 – Функція виведення матриці на консоль

Для введення матриці з консолі можна скористатися функцією, що наведена на рисунку 9.3.

У цій функції матриця вводиться рядками, але особливість функції полягає у тому, що у разі неоднакової кількості елементів у рядках, що вводяться, кількість елементів у кожному рядку матриці буде дорівнювати кількості елементів останнього рядка. Зайві елементи інших рядків будуть ігноруватися, а ті що не вводилися будуть дорівнювати 0, бо уся ділянка пам’яті, що відведена для матриці перед початком вводу заповнюється нулями.

Для перетворення рядка символів у масив використовується функція strToArr, розглядалася у лабораторній роботі №7.

```
void getMatrixFromConsole(int matr[][COLS], int &nRow, int &nCol){
   char s[80];   //Масив символів для введення рядка матриці
   int ar[COLS]; // Масив чисел для поточного рядка матриці
   //Обнуляємо ділянку пам'яті для матриці
   for(int i = 0; i < ROWS; i++)
       for(int j = 0; j < COLS; j++)
          matr[i][j]=0;
   cout<<"Введіть кількість рядкв матриці: ";
   cin>>nRow;
   gets(s);//Забираємо "Enter", що не забрав cin
   for( int i=0; i<=nRow; i++){
      system("cls"); //чистимо екран
      cout<<"Усього рядків у матриці "<<nRow<<"\n";
      if (i>0){
         cout<<"Введені рядки матриці:";
         printMatrix(matr, i, nCol);
      }
      if(i<nRow){
         cout<<"Введіть рядок № "<<i<<endl;
         gets(s);//Читаємо рядок символів
         strToArr(s,ar,nCol); //Перетворюємо символи у масив чисел
         // Записуємо у матрицю черговий рядок
         for(int k=0; k<nCol;k++){
            matr[i][k]=ar[k];
         }
      }
   }
}
```

Рисунок 9.3 – Функція введення матриці з консолі

9.1.4 *Тотальна обробка даних у матрицях*

Тотальна обробка передбачає виконання однакових операцій для всіх елементів матриці.

Нижче перераховані деякі завдання тотальної обробки матриць:

- заповнення матриці випадковими або іншими числами;
- пошук суми всіх елементів матриці;
- пошук максимального або мінімального елемента в матриці;
- множення матриці на число;
- отримання суми двох матриць однакового розміру.

Тотальна обробка зазвичай реалізується за допомогою двох вкладених циклів for, параметрами яких є індекси матриці. Якщо матриця обробляється по рядках, то заголовок зовнішнього циклу записується для першого індексу, а якщо по стовпцях, то перший індекс повинен змінюватися у внутрішньому циклі.

Прикладом тотальної обробки матриці може служити функція заповнення елементів числової матриці нулями, що наведена нижче.

До функції передається матриця із зазначенням заявленої кількості елементів у рядку, а також кількість рядків (nRow) та кількість елементів у рядку (кількість стовпців nCol), що підлягають опрацюванню.

Перед описом функції наведені директиви #define, що визначають максимально можливий розмір матриці.

Всі завдання тотальної обробки мають подібну структуру. Відрізняються вони тільки інструкціями всередині циклу.

```
#define ROWS 10
#define COLS 8
…
void matrixToZero(int matr[][COLS], int nRow, int nCol){
   for(int i = 0; i < row; i++)
      for(int j = 0; j < col; j++)
        matr[i][j]=0;
}
```

9.1.5 *Вибіркова обробка матриць*

При вибірковій обробці матриці можуть вирішуватися ті ж завдання що і при тотальній, але тільки для частини елементів матриці. Ось приклади можливих варіантів групування елементів матриці:

- елементи головної діагоналі квадратної матриці;
- елементи допоміжної діагоналі квадратної матриці;
- елементи деякого стовпця або рядка матриці;
- елементи матриці, розташовані по її кромці;

– елементи квадратної матриці, розташування яких відповідає розташуванню білих або чорних клітин шахової дошки.

Для перших трьох з перерахованих вище варіантів обробка здійснюється в одному циклі, бо під час перебору елементів тільки один індекс є незалежним.

У першому з наведених варіантів у оброблюваних елементів, які розташовані на головній діагоналі матриці, номери рядка і стовпця співпадають. Тому в циклі можна змінювати номер стовпця, а рядку привласнювати той же самий номер.

У другому випадку, на допоміжній діагоналі, номер стовпця j зв'язаний з номером рядка і співвідношенням j = nCol -1-i. У цьому співвідношенні nCol - це кількість елементів у рядку (кількість стовпців).

У разі обробки одного із стовпців матриці, його номер фіксований, і єдиний цикл достатньо організувати тільки за елементами рядка. Аналогічно оброблюється і окремий рядок.

При обробці елементів матриці розташованих по її кромках можна організувати чотири цикли, що виконуються послідовно один за одним. У кожному циклі обробляються елементи чергового рядка або стовпця, що утворюють кромку матриці. Обробляти слід всі елементи за винятком того, який буде початком наступної послідовності, тобто останнього або першого.

При «шаховому» групуванні, можна організувати подвійний цикл, як при тотальній обробці, але обробляти тільки ті елементи, сума індексів яких є парною (або непарною).

9.1.6 *Перестановки елементів матриці*

Є завдання, де потрібно міняти місцями елементи матриці. Нижче перераховані деякі з таких завдань:

– транспонування матриці (поворот навколо головної діагоналі);
– поворот навколо допоміжної діагоналі;
– поворот навколо горизонтальної осі;
– поворот навколо вертикальної осі.

Для вирішення всіх цих завдань потрібен подвійний цикл. Зовнішній цикл зазвичай перебирає усі номери рядків або стовпців, іноді, за винятком першого чи останнього номера. Внутрішній же цикл забезпечує опрацювання тільки половини елементів, які розташовані з одного боку від осі повороту. Тіло циклу містить оператори, які забезпечують обмін значеннями між поточним елементом матриці і симетричним йому. Для організації обміну найзручніше використовувати проміжну змінну.

Основні труднощі, що виникають при вирішенні цих завдань, це визначення індексів елемента, симетричного поточному елементу.

У таблиці 9.1 наведені параметри циклів, що забезпечують перевороти елементів матриці навколо різних осей симетрії. Індексні вирази, наведені в таблиці, складені в припущенні, що нижні індекси рядків і стовпців рани одиниці, верхні індекси стовпців і рядків відповідно рівні nCol і nRow, а m - ім'я матриці.

Таблиця 9.1 – Параметри циклів при перестановках елементів матриці

Лінія симетрії у повороті	Зовнішній цикл, за рядками (індекси рядка)	Внутрішній цикл, за елементами рядка (індекси стовпця)	Індекси елементів	
			поточні	симетричні
Головна діагональ	від i=1; i < n	від j = 0; j< i	[i, j]	[j, i]
Допоміжна діагональ	від i=0 ; i < n -1	від j=0 ; j<n -1- i	[i, j]	[n-1-j, n-1-i]
Горизонтальна вісь	від i=0; i<nRow/2	від j=0; j< nCol	[i, j]	[nRow -1–i, j]
Вертикальна вісь	від i=0; i<nRow	від j = 0; j< nCol / 2	[i, j]	[i, nCol -1- j]

9.1.7 *Видалення та вставка елементів матриці*

Тут мається на увазі видалення стовпця або рядка матриці, бо видалити окремий елемент матриці ми не можемо, не порушивши прямокутну структуру.

Видалення або вставка рядка чи стовпця подібно видаленню або вставці елементу масиву. При видаленні потрібно зрушити наступні рядки вліво або стовпці вгору, на місце видаленого компонента, і зменшити відповідний розмір. При вставці потрібно зрушити наступні рядки вправо або стовпці вниз, звільнивши місце для компоненту що вставляється, і збільшити відповідний розмір матриці.

9.1.8 *Сортування елементів матриці*

Є два різновиди задачі сортування матриць.

Перший з них полягає у тому, що сортуються окремі частини матриці, незалежно від інших частин. До цього різновиду можна віднести наступні задачі:

– сортування кожного рядка або кожного стовпця незалежно від інших;
– сортування однієї з діагоналей матриці;
– сортування рядків або стовпців матриці за значенням першого елементу групи;
– сортування рядків матриці по сумі елементів рядка;
– сортування стовпців за сумою елементів стовпця.

Вирішення цих завдань не набагато складніше сортування масиву.

Для вирішення першого завдання із наведеного списку, сортування масиву слід застосувати послідовно для кожного рядка або стовпця.

У другій задачі діагональ розглядається як одновимірний масив, і важливо тільки не заплутатися в індексах, якщо потрібно відсортувати допоміжну діагональ.

При вирішенні третього завдання додаткова проблема виникає тільки в тому, що потрібно міняти місцями цілі рядки або стовпці. Для цього можна написати окрему функцію.

Для вирішення останнього завдання потрібно буде ще написати функцію обчислення суми елементів стовпця або рядка.

Другий різновид завдань сортування матриць полягає у тому, що в одній процедурі сортування беруть участь всі елементи матриці

Таке сортування забезпечує розташування елементів матриці за зростанням або спаданням в напрямку заданого способу обходу матриці.

Варіанти обходу елементів матриці можуть бути найрізноманітніші, але більшість з них має тільки навчальну користь. Деякі з них наведені нижче:

– по рядках зверху вниз і зліва направо, або знизу вгору і справа наліво, або змійкою;

– те ж саме по стовпцях;

– те ж саме по лініях паралельним головній або допоміжній діагоналях;

– кутом, вершина якого знаходиться на головній або допоміжній діагоналі.

Яким би не був спосіб обходу елементів матриці, саму процедуру сортування завжди можна звести до сортування масиву. Для цього слід переписати елементи матриці в масив. Потім масив відсортувати одним з відомих способів. Після цього елементи масиву потрібно знову переписати в матрицю, обходячи її за відповідним маршрутом.

Написати функцію переписування матриці в масив не складає труднощів, ця функція наведена на рисунку 9.4.

```
void matrixToArr(int matr[][COLS], int ar[], int nRow, int nCol){
   int n = 0;
   for(int i = 0; i< nRow; i++)
      for( int j = 0; j<nCol; j++)
         ar[n++]=matr[i][j];
}
```

Рисунок 9.4 – Функція переписування матриці у масив

Проблема сортування масиву нами теж вже вирішена. Можна використовувати будь-який з раніше розглянутих методів.

Таким чином, проблема сортування матриці зводиться до написання процедури, що забезпечує обхід елементів матриці у відповідності з необхідним маршрутом.

Підходи до створення процедур маршрутизації можуть бути різними.

Для простих варіантів сортування, коли напрям сортування збігається зі стовпчиками або рядками, можна використовувати розрахунковий спосіб визначення координат необхідного елемента.

Як приклад використання розрахункового способу, розглянемо маршрут обходу матриці по рядках зверху вниз і справа наліво для всіх рядків.

Припустимо, що матриця має 3 рядки і 4 стовпця, і обхід матриці

відбувається по рядках зліва направо. Тоді елементи матриці умовно слід перенумерувати так, як показано на рисунку 9.5.

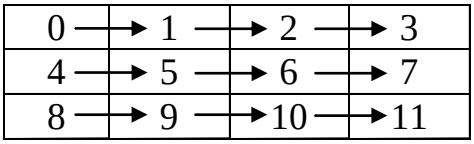


Рисунок 9.5 – Схема маршруту обходу матриці при сортуванні по рядках

Нехай номер елементу в масиві дорівнює 7. Координати цього елемента в матриці будуть такими.

Номер рядка буде 7 / 4 = 1.

Номер стовпця буде 7 % 4 = 3.

На рисунку 9.6 наведено функцію, що реалізує запис елементів масиву в матрицю і використовує наведені розрахункові формули.

Вихідними даними для процедури є:

– matr - матриця, яку потрібно заповнити по рядках;
– nCol - кількість стовпців матриці;
– nRow - кількість рядків матриці;
– ar - упорядкований масив, з якого беруться дані заповнення матриці.

У функції обчислюються номер рядка “r” та номер стовпця “c” матриці, що відповідають номеру “i” елемента в масиві, і у відповідності зі значеннями цих координат елемент масиву переписується в матрицю.

```
void fillMatrFromArHrzLine(int matr[][COLS], int nRow, int nCol, int ar[] ){
   for(int i = 0; i < nRow*nCol; i++){
      int r = i / nCol;
      int c = i % nCol;
      matr[r][c] = ar[i];
   }
}
```

Рисунок 9.6 – Функція переписування впорядкованого масиву до матриці по рядкам

У тих випадках, коли маршрут сортування складний, розрахунковий спосіб непридатний.

Для реалізації складних маршрутів сортування можна рекомендувати метод моделювання маршруту. Суть цього методу, полягає у створенні алгоритму, який для визначення номера рядка і стовпця моделює проходження маршруту сортування від початку до останнього елемента.

Алгоритм маршрутизації на кожному кроці повинен оцінювати положення поточного елемента і визначати, виходячи з цього, координати

наступного. Таким чином, в процесі роботи алгоритму послідовно обчислюються координати всіх елементів маршруту, поки не буде досягнутий останній.

В якості прикладу можна навести процедуру маршрутизації для сортування квадратної матриці «Кутом, зверху - вниз - ліворуч, від початку головної діагоналі». Схема маршруту реалізованого в процедурі наведена на рисунку 9.7.

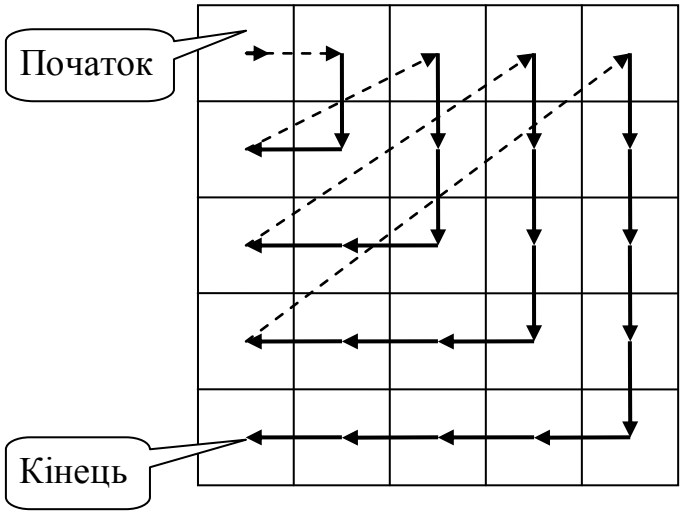


Рисунок 9.7 – Схема маршруту обходу матриці кутом униз і ліворуч, у напрямку головної діагоналі

Наведена схема реалізована у функції, що зображена на рисунку 9.8.

```
void fillMatrFromArCornerTopLeft(int matr[][COLS],int nRow, int nCol,int ar[]){
   int r=0, c=0; // Координати початку маршруту
   for(int i=0; i<nRow*nCol; i++){
      matr[r][c]=ar[i];
      if(c == 0){
         c = r + 1;
         r = 0;
      }
      else if( r < c)  r++;
      else c--;
   }
}
```

Рисунок 9.8 – Функція реалізації маршруту обходу матриці кутом униз і ліворуч, у напрямку головної діагоналі

Назви формальних параметрів у цій функції такі ж, як і в попередньому прикладі.

У тілі функції організовано цикл послідовного формування номерів стовпця і рядка для елементів відсортованого масиву. При кожному повторенні циклу обчислюються координати чергової точки маршруту. Спосіб обчислення координат наступної точки змінюється при досягненні лівого краю матриці або головної діагоналі.

Якщо поточний елемент знаходиться у лівому стовпцю, то наступний елемент потрапляє на перший рядок.

При переході через головну діагональ змінюється правило зміни координат елементів матриці. Вище діагоналі збільшується на одиницю номер рядка (рух униз). Нижче діагоналі слід зменшувати на одиницю номер стовпця (рух ліворуч).

9.2 ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

В лабораторній роботі слід створити програму, відповідно до вимог варіантів з таблиць 9.2, 9.3, 9.4 и 9.5.. Номер варіанту вибирається відповідно до останньої цифри номеру залікової книжки.

Інтерфейс програми має забезпечити тестування ваших завдань для довільних вхідних даних.

Таблиця 9.2 Задачі тотальної обробки матриць

Варіант	Завдання
0	Знайти найбільше і найменше число та його координати в матриці випадкових чисел.
1	Підрахувати кількість нулів і одиниць в матриці, що складається з випадкових двійкових чисел.
2	Знайти найбільше і найменше з парних чисел та їх координати в матриці.
3	Знайти в матриці координати усіх елементів, що дорівнюють заданому числу.
4	Підрахувати суми для парних і непарних чисел в матриці.
5	Порівняти попарно елементи двох матриць та створити третю, елементи якої дорівнюють більшому числу з кожної пари.
6	Створити матрицю, в якій нулі розташовані в клітинах з парною сумою індексів. Решту клітини заповнити одиницями.
7	З матриці, що заповнена числами, створити нову матрицю, елементи якої дорівнюють сумі цифр чисел у вихідній матриці.
8	Створити матрицю, значення елементів якої дорівнюють сумі індексів цих елементів.
9	Підрахувати, скільки разів зустрічається задане число в матриці.

Таблиця 9.3 Задачі на вибіркову обробку матриць

Варіант	Завдання
0	Створити масив, елементи якого дорівнюють кількості цифр чисел, розташованих по кромці матриці.
1	Створити масив, елементи якого дорівнюють максимальним елементам в непарних стовпцях матриці.
2	Створити масив, елементи якого дорівнюють мінімальним елементам в парних рядках матриці.
3	Створити масив, елементи якого дорівнюють сумі цифр чисел, розташованих по кромці випадково заповненої матриці.
4	Створити масив, елементи якого дорівнюють сумам пар чисел, розташованих на головній і допоміжної діагоналі матриці.
5	Створити масив, елементи якого відповідають заданому стовпцю матриці.
6	Створити масив, елементи якого дорівнюють сумі елементів в непарних стовпцях матриці.
7	Створити масив, елементи якого дорівнюють сумі елементів у парних рядках матриці.
8	Створити масив, елементи якого відповідають заданому рядку матриці.
9	Створити масив, елементи якого дорівнюють сумам пар чисел, на осях квадратної матриці непарного розміру

Таблиця 9.4 Задачі на перестановку елементів матриці

Варіант	Завдання
0	Поміняти місцями найбільший та найменший елементи матриці.
1	Перевернути квадратну матрицю навколо другої діагоналі
2	Перевернути матрицю навколо горизонтальної осі.
3	Перевернути матрицю навколо вертикальної осі.
4	Поміняти місцями елементи головної і допоміжної діагоналі матриці.
5	Поміняти місцями елементи вертикальної та горизонтальної осі квадратної матриці з непарним розміром.
6	Перевернути задом - наперед елементи головної діагоналі квадратної матриці.
7	Перевернути задом - наперед елементи допоміжної діагоналі квадратної матриці.
8	Перемістити елементи по кромці квадратної матриці так, щоб перший рядок став останнім стовпцем, останній стовпець - нижнім рядком у зворотному порядку, нижній рядок - першим стовпцем і перевернутий перший стовпець - першим рядком.
9	Видалити задані стовпець і рядок матриці

Таблиця 9.5 Задачі на сортування матриці

Варіант	Маршрути сортування кутом
0	Згори – донизу – ліворуч, з кінця головної діагоналі.
3	Зліва – праворуч – догори, від початку головної діагоналі.
2	Зліва – праворуч – догори, з кінця головної діагоналі.
3	Знизу – верх – направо, від початку головної діагоналі.
4	Знизу – верх – праворуч, з кінця головної діагоналі.
5	З правого боку – ліворуч – униз, від початку головної діагоналі.
6	З правого боку – ліворуч – униз, з кінця головної діагоналі.
7	Зліва – праворуч – вниз, від початку допоміжної діагоналі
8	Згори – донизу – праворуч, з кінця допоміжної діагоналі.
9	Зліва – праворуч – униз, з кінця допоміжної діагоналі.

9.3 ВИМОГИ ДО ЗВІТУ

- Назва роботи.
- Мета роботи.
- Способи визначення та ініціалізації матриць.
- Матриці – як параметри функцій.
- Особливості сортування матриць.
- Схема алгоритму маршрутизації для свого варіанту сортування.
- Тексти функцій для індивідуальних завдань з коментарями.
- Результати тестування проекту у вигляді копій консолі.
- Висновки.

9.4 КОНТРОЛЬНІ ПИТАННЯ

- Способи визначення та ініціалізації матриць.
- Матриці – як параметри функцій.
- Тотальна обробка даних у матрицях. Приклади.
- Вибіркова обробка матриць. Приклади..
- Перестановки елементів матриці. Приклади.
- Видалення та вставки елементів матриць. Приклади.
- Особливості сортування матриць.
- Написати функцію для обробки матриці за вказівкою викладача.
- Написать функцію відповідно до одного із варіантів індивідуальних завдань до лабораторної роботи.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программирования С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с
2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. –1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вильямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

10 ЛАБОРАТОРНА РОБОТА № 10. РОБОТА ІЗ СТРУКТУРАМИ

Мета роботи:
- Ознайомитися із поняттям структура.
- Ознайомитися із способами оголошення та ініціалізації структур.
- Опанувати алгоритми обробки структур.

10.1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Структура - це тип даних, якому відповідає суміш елементів даних різних типів. Кожен з таких елементів, що входять до складу структури, називають полем. При роботі зі структурами оперують поняттями ім'я структури і ім'я поля. Імена структурам та їх полям присвоюються у відповідності зі стандартними правилами конструювання імен ідентифікаторів. Програміст може оперувати як з усією структурою, так і з окремими полями - це залежить від розв'язуваної задачі і операторів, що використовуються.

Для структури використовується її ім'я, а для ідентифікації її складових частин використовується складене ім'я, яке складається з імені структури та імені поля, розділених крапкою.

Для конструювання структури використовується поняття шаблону структури.

10.1.1 *Оголошення шаблону та ініціалізація структур*

Синтаксис оголошення шаблону структури виглядає так, як показано на рисунку 10.1.

```
struct
        < тег шаблону структури >{
                <тип поля1>  <ім’я поля1>;
                < тип поля2>  <ім’я поля2>;
                 ...
                <тип поляN>  <ім’я поляN>;
        };
```

Рисунок 10.1 – Синтаксис оголошення шаблону структури

На цьому рисунку:

– struct – службове слово, що використовується для визначення структур;

– тег шаблону структури – ім’я, яким позначають структури даної конструкції, фактично відіграє роль типу структури, хоча поняття тип не використовується;

– { } – дужки, що обмежують перелік полів структури;

– тип поля та ім’я поля – стандартне оголошення кожного з полів.

В якості прикладу розглянемо шаблон структури для збереження результатів атестації студентів. Структура буде містити чотири поля - прізвище студента з ініціалами, назва групи, в якій навчається студент, середній бал поточної успішності та кількість незадовільних оцінок.

Нижче наведено цей шаблон.

```
struct stud {
   char fio[20];
    char gr[10];
    float srBall;
    int nzd;
};
```

Оголосити змінні, що мають відповідну структуру можна так:

```
struct stud s1, s2, s3;
```

Слово struct у оголошенні таких змінних використовувати не обов’язково:

```
stud s1, s2, s3;
```

Змінні можна оголошувати і одночасно із оголошенням шаблону:

```
struct stud {
   char fio[20];
    char gr[10];
    float srBall;
    int nzd;
} s1, s2, s3;
```

При оголошенні структур їх можна ініціалізувати. Список полів обмежується фігурними дужками, а поля розділяються комами:

```
stud  bestStud = {Гетьман Т.Г., КІ-131, 5.00, 0};
```

Значення полів можна визначати за допомогою оператора присвоєння:

```
s1.fio =’Петренко А.П.’;
s1.gr = ‘КС041’;
s1.srBall = ‘3.2’;
s1.nzd = ‘1’;
```

Оператор присвоювання можна використовувати і для запису в цілому. У

прикладі, наведеному нижче, всім полям запису s2 будуть присвоєні значення полів запису s1.

```
s2 = s1;
```

Полями структури можуть бути також і структури.

10.1.2 *Масиви структур*

Із структур однакового тегу може бути створений масив, точно так само як з даних інших типів. Наприклад, якщо ми хочемо мати результати атестації для всіх студентів, можна створити масив, що містить дані про кожного студента у вигляді структури і частково його проініціалізувати:

```
#define FIO_SIZE 15
#define GR_SIZE 7
#define AR_SIZE 30
struct stud {char fio[FIO_SIZE];
        char gr[GR_SIZE];
        float srBall;
        int nzd;
};
stud ar[AR_SIZE]={
            {"Чуб П.П.","КС051", 4.55, 0},
            {"Гай А.Л.","КС052", 1.55, 3},
            {"Кит А.В.","КС051", 2.45, 1},
            {"Кіт С.В.","КС052", 1.25, 3},
            {"Бут К.Л.","КС052", 4.65, 0}
};
int size=5;
```

При зверненні до елементу масиву структур також використовуються індекси та квадратні дужки. При цьому слід враховувати, що елементом масиву є структура і, тому, квадратні дужки ставляться після імені масиву, а потім, через крапку, записується ім'я поля.

Наприклад, вище наведений масив можна доповнити ще однією структурою.

```
ar[size].fio =’Петренко А,П,’;
ar[size].gr = ‘КС041’;
ar[size].srBall = ‘3.2’;
ar[size].nzd = 1;
```

Якщо полем структури є масив, то у зверненні до елементу масиву квадратні дужки вже будуть після імені поля. Так, наприклад, якщо ми хочемо у вищенаведеному масиві структур змінити прізвище студента «Гай» на «Гак», то слід написати так:

```
ar[1].fio[2] = 'к';
```

10.1.3 *Введення-виведення структур*

При організації вводу-виводу варто мати на увазі, що безпосередньо структуру ввести або вивести, без використання спеціально написаних процедур, не можна. Можна вводити або виводити окремі поля структури, або елементи полів, якщо поля є теж структурами.

Введення і виведення структур схоже на введення та виведення матриць, але різниця тут у тому, що стовпчики можуть мати різні типи, і доступ до елементів різних стовпців проводиться не за індексом, а за назвою.

10.1.4 *Сортування масивів структур*

Сортування масиву структур проводиться так само, як і сортування масиву чисел, тобто структури порівнюються і, якщо необхідно, переставляються. Єдина проблема тут полягає в тому, що способів порівняння записів, а, отже, і варіантів сортування, може бути багато, і кожен з них може знадобитися. Так для масиву, розглянутого в попередньому прикладі, записи можна сортувати за прізвищем студента або за середнім балом, за кількістю незадовільних оцінок і т.д. Більш того, можливі і складніші випадки сортування, наприклад, по групі, а в межах однієї групи на прізвище студента. У такій ситуації доводиться для кожного варіанту сортування писати функції сортування, які будуть відрізнятися тільки способом порівняння структур.

10.2 СТВОРЕННЯ ПРОЕКТУ «РЕЗУЛЬТАТИ АТЕСТАЦІЇ»

У цьому проекті ми будемо працювати з масивом, що містить результати атестації студентів. Кожен елемент масиву буде зберігати запис, полями якої будуть прізвище студента з ініціалами, найменування групи, в якій навчається студент, середній бал і кількість незадовільних оцінок у студента.

Потрібно забезпечити введення потрібної кількості даних до масиву структур, сортування масиву за різними правилами та виведення результатів обробки масиву.

У проекті буде реалізовано такі процедури обробки даних.

- комплексне сортування по групі + прізвище студента.
- комплексне сортування за кількістю незадовільних оцінок ↓ + та середньому балу ↑.
- вибірка студентів, які мають середній бал вище заданого.
- підрахунок загального числа студентів, що мають більш 2-х незадовільних оцінок.

Даний проект може слугувати прикладом того, як виконати завдання для

самостійної роботи.

10.2.1 *Інтерфейс користувача для проекту*

Можливий варіант інтерфейсу користувача представлено на рисунку 10.2.

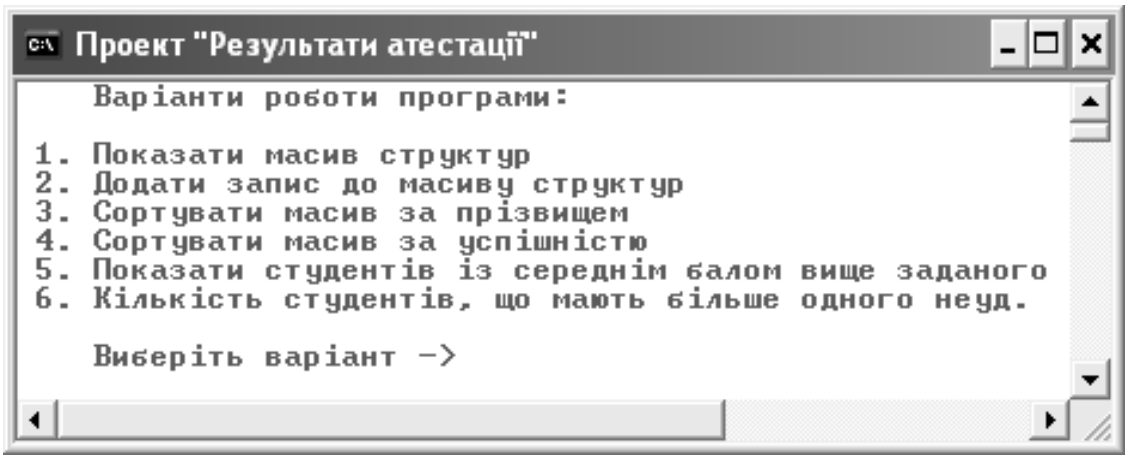


Рисунок 10.2 – Інтерфейс користувача проекту

Формування інтерфейсу користувача можна доручити головній функції програми, функції main(). Виведення варіантів роботи програми і введення номеру вибраного варіанту доцільно реалізувати у безкінечному циклі з очисткою екрану перед виведенням варіантів.

Нижче наведено фрагмент цієї функції:

```
SetConsoleTitleA("Проект \"Результати атестації\"");
int v;
while (true) {
   system("cls");
   cout<<"    Варіанти роботи програми:\n\n";
   cout<<" 1. Показати масив структур\n";
   . . .
   cout<<"\n    Виберіть варіант -> ";
   cin>>v;
   operate_chois(v);
};
```

Нагадаємо, що для того, щоб отримати доступ до функції system(), потрібно підключити заголовний файл <windows.h>.

10.2.2 *Визначення глобальних типів даних програми*

Для реалізації проекту потрібно оголосити шаблон для структури, масив для зберігання інформації про студентів, розміри полів для прізвища, назви групи, розмір масиву. Можна також ініціалізувати декілька елементів масиву і

оголосити змінну, яка буде зберігати реальну кількість елементів у масиві.

Ці оголошення повинні бути доступні усім функціям програми, тому їх слід зробити поза цими функціями, на початку програми.

Тексти цих оголошень і ініціалізація масиву наведені нижче:

```
#define FIO_SIZE 15
#define GR_SIZE   7
#define AR_SIZE 30
struct stud {char fio[FIO_SIZE];
          char gr[GR_SIZE];
          float srBall;
          int neud;
};
stud ar[AR_SIZE]={
              {"Чуб П.П.","КС051", 4.55, 0},
              {"Гай А.Л.","КС052", 1.55, 3},
              {"Кит А.В.","КС051", 2.45, 1},
              {"Кіт С.В.","КС052", 1.25, 3},
              {"Бут К.Л.","КС052", 4.65, 0}
};
int size=5;
```

10.2.3 *Функція обробки номеру вибраного варіанту*

Із фрагменту функції main(), наведеного у пункті 10.2.1, видно, що для обробки номеру вибраного варіанту використовується функція operate_chois().. Номер вибраного варіанту у цій функції опрацьовується за допомогою оператора swith. Текст цієї функції ми тут не наводимо, бо вона подібно до функцій обробки номеру варіанта з інших лабораторних робіт.

10.2.4 *Функція відображення масиву на консолі*

Алгоритм функції полягає в циклічному виведенні полів кожного з елементів масиву структур. Для форматування виводу використовуються функції setw(), setprecision(). Зауважимо, що використання цих функцій можливо при умові включення заголовного файлу <iomanip>.

Нижче наводиться код цієї функції

```
void showAr(){
   for(int i=0;i<size;i++){
      cout<<setw(16)<<ar[i].fio
            <<setw(8)<<ar[i].gr
            <<setw(3)<<ar[i].neud
            <<setw(6)<<setprecision(3)<<ar[i].srBall<<endl;
   }
}
```

10.2.5 *Додавання нових даних до масиву структур*

Додати нового студенту у масив структур дуже просто. Для цього достатньо ввести з консолі значення кожного поля наступного елементу масиву і збільшити лічильник елементів масиву на 1.

Код відповідної функції наведено нижче:

```
void addStud(){
  cout<<"Введіть прізвище: ";
  gets(ar[size].fio); //Пропускаємо попередній символ кінця рядка
  gets(ar[size].fio);
  cout<<"Введіть групу: ";
  cin>>ar[size].gr;
  cout<<"Введіть середній бал і кількість заборгованостей: ";
  cin>>ar[size].srBall>>ar[size].neud;
  size++;
  cout<<endl;
  cout<<"Таким став масив:\n";
  showAr();
};
```

У наведеній функції для введення прізвища з ініціалами, де є пробіли, використовується функція gets(). Але для цього необхідно підключити заголовний файл <cstdio> або <stdio.h>

10.2.6 *Функція сортування масиву за групою та прізвищем*

Особливість сортування масивів структур полягає в тому, що може бути дуже багато варіантів сортування. Можливі сортування за значеннями окремих полів, а також по їх комбінаціям. Наприклад, можна сортувати масив структур за кількістю «хвостів», а при рівності цього показника по середньому балу, крім того, при рівності середнього балу за прізвищами.

Алгоритм сортування для кожного з можливих варіантів залишається одним і тим же, змінюється тільки правило порівняння записів. У тих випадках, коли алгоритм порівняння записів складний, його доцільно представити у вигляді окремої функції. Виходячи з того, що стандартні функції порівняння масивів символів, які доводиться використовувати для порівняння структур, повертають цілочислове значення, доцільно у всіх функціях для порівняння структур повертати результат цілочислового типу. Цей результат буде від'ємний, якщо перший параметр менший, додатній - якщо перший параметр більший, та нуль, якщо параметри однакові.

Наприклад, при необхідності сортувати масив записів по групі і

прізвища, можна попередньо написати таку функцію порівняння:

```
int cmp1(stud s1, stud s2){
   if(strcmp(s1.gr,s2.gr)!=0)
      return strcmp(s1.gr,s2.gr);
   return strcmp(s1.fio,s2.fio);
}
```

Після цього можна написати процедуру сортування масиву структур, наприклад, методом вибору, яка відрізнятиметься від тих, що розглядалися раніше, тільки тим, що для порівняння двох елементів масиву викликається написана вище функція.

```
void sort1(){
   for(int i=0; i<size-1;i++){
      for(int j = i + 1; j<size; j++) {
         if ((cmp1(ar[i],ar[j])>0)){
            stud x =  ar[i];
            ar[i] =  ar[j];
            ar[j] =  x;
         }
      }
   }
}
```

10.2.7 *Сортування за кількістю незадовільних оцінок та середньому балу*

Для цього сортування можна також попередньо написати відповідну функцію порівняння. Нижче наведена така функція:

```
int cmp2(stud s1, stud s2){
   if(s1.neud!=s2.neud)
      return s2.neud-s1.neud;
   if(s1.srBall-s2.srBall>0)
      return 1;
   return -1;
```

Наявність цієї функції дозволяє нам написати функцію сортування, яка буде відрізнятися від розглянутої вище тільки ім'ям функції, що використовується для порівняння. Замість виклику cmp1(ar [i], ar [j]), у

процедурі сортування потрібно буде написати cmp2 (ar [i], ar [j]). Крім того, у функції повинно бути інше ім'я, наприклад, sort2.

10.2.8 *Вибірка студентів, що мають середній бал вище 4*

Вирішення цієї задачі мало чим відрізняється від задачі виведення масиву на консоль. Різниця полягає у тому, що виводити потрібно тільки ті структури, що відповідають заданій умові. Окрім того, масив перед обробкою доцільно відсортувати.

10.2.9 *Підрахунок кількості студентів що мають більше 2-х незадовільних оцінок*

Вирішення цієї задачі потребує підрахунку кількості елементів у масиві, що відповідають заданій умові. Задача відома з попередніх робіт і вирішується достатньо просто.

10.3 ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

В лабораторній роботі слід створити програму, у якій створюється та обробляється масив структур відповідно до вимог варіанту з таблиці 10.1. Номер варіанту вибирається відповідно до останньої цифри номера залікової книжки.

Таблиця 10.1 – Завдання для самостійної роботи

№	Зміст структур проекту та вимоги до обробки
0	Номер кімнати у гуртожитку, кількість комп'ютерів, наявність холодильника, кількість мешканців. Сортування обміном за номером кімнати. Комплексне сортування за кількістю мешканців ↓ + за кількістю комп'ютерів ↓ + по наявності холодильника ↓. Вибірка кімнат, де на кожного мешканця не менше 1-го комп'ютера. Підрахунок загальної кількості електроприладів
1	Прізвище працівника, оклад, надбавка в цілих %, премія. Сортування вибором по прізвищу. Комплексне сортування за сумарною платнею ↑ + премією ↑ + за прізвищем ↓. Вибірка осіб, у яких сумарна виплата вище заданого значення. Підрахунок загальної суми виплат.
2	Прізвище студента, дата (у вигляді рядка символів рр-мм-дд), витрати на сніданок, обід, вечерю в студентській їдальні. Комплексне сортування вставкою по даті ↓ + прізвище ↓. Комплексне сортування за прізвищем ↓ + по сумі витрат за день ↑. Вибірка даних про щоденні сумарні витрати, для заданого студента. Підрахунок сумарних витрат студента за місяць.

Продовження таблиці 10.1

3	Прізвище студента, група, контракт чи бюджет, середній бал. Комплексне сортування обміном по групі ↓ + на прізвище ↓. Комплексне сортування по полю контракт або бюджет + за середнім балом. Вибірка студентів, у яких середній бал нижче заданого. Обчислення середнього балу для заданої групи.
4	Номер кімнати в гуртожитку, площа, число мешканців, балкон. Сортування вибором по номеру кімнати. Комплексне сортування за наявністю балкона + по площі, що припадає на одного мешканця. Вибірка кімнат, де площа на одного мешканця менше норми. Підрахунок кількості мешканців у гуртожитку.
5	Номер корпусу, номер аудиторії, кількості місць, наявність ТСО. Комплексне сортування вставкою за номером корпуса + номер аудиторії. Комплексне сортування за наявністю ТСО + кількість місць. Вибірка аудиторій з ТСО з числом місць не менше заданого. Підрахунок загальної кількості місць в заданому корпусі.
6	Прізвище спортсмена, результат на 100-метрівці, результат зі стрибків у довжину, результат зі стрибків у висоту. Сортування обміном по кожному виду і визначення місця спортсмена в даному виді. Сортування по сумі місць у всіх видах. Вибір трійки кращих бігунів на 100 метрів. Середній результат спортсменів на 100-метрівці.
7	Дата (у вигляді рядка символів рр-мм-дд), температура повітря вдень, атмосферний тиск, вологість. Сортування вибором по даті. Комплексне сортування за температурі + по тиску + по вологості. Визначення середньої температури заданого місяця. Знайти дні місяця, коли температура перевищувала середнє значення.
8	Прізвище хворого, номер палати, температура, тиск, пульс. Сортування вставкою за прізвищем. Комплексне сортування по палаті + по температурі + по тиску + по частоті пульсу. Підрахунок числа хворих з температурою вище заданої. Виведення списку хворих з тиском вище допустимого.
9	Група, прізвище старости, кількість контрактників, бюджетників. Сортування обміном по групах. Комплексне сортування за загальною кількістю студентів в групі + за кількістю бюджетників. Формування переліку груп із загальною кількістю студентів. Підрахунок загальної кількості контрактників та бюджетників.

10.4 ВИМОГИ ДО ЗВІТУ

– Назва роботи.
– Мета роботи.
– Оголошення та ініціалізація структури.
– Інтерфейс проекту.
– Тексти функцій для індивідуальних завдань з коментарями.
– Результати тестування проекту у вигляді копій консолі.
– Висновки.

10.5 КОНТРОЛЬНІ ПИТАННЯ

– Оголошення структури та її ініціалізація.
– Особливості масивів структур та їх ініціалізації.
– Структури як параметри функцій.
– Оголосити та ініціалізувати структуру за вказівкою викладача.
– Оголосити та ініціалізувати структуру відповідно до одного із варіантів індивідуальних завдань до лабораторної роботи.
– Написати функцію для обробки масиву структур.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программирования С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с
2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. – 1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вильямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

11 ЛАБОРАТОРНА РОБОТА № 11. ОСНОВИ РОБОТИ З ПОКАЖЧИКАМИ

Мета роботи:
- Ознайомитися з поняттям покажчик.
- Навчитися оголошувати та ініціалізувати покажчики.

11.1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Покажчик або вказівник – це особливий тип даних, значенням якого є адреса певного байту оперативної пам'яті. Цей тип даних найчастіше використовується у системному програмуванні, але і прикладні програмісти успішно використовують цей інструмент. Слід сказати, що не усі мови програмування надають можливість використовувати покажчики, але у таких популярних мовах програмування як С та Паскаль робота з покажчиками можлива. Більш того, можна сказати, що покажчики – це візитна картка мови С.

Вважається, що використання покажчиків дає змогу скоротити текст програми та підвищити її ефективність. Але тут є і зворотна сторона. Покажчики – це не зовсім простий у використанні засіб програмування і часто стають джерелом помилок, виявити які у програмі дуже складно. Тому у деяких мовах прикладного програмування (наприклад, Java) принципово відмовилися від покажчиків.

11.1.1 *Оголошення та ініціалізація покажчиків*

Оголошення вказівника має вигляд, представлений на рисунку 11.1.

<тип даних> *<ім'я покажчика> ;

Рисунок 11.1 – Синтаксис оголошення покажчика

У цьому оголошенні тип даних визначає тип елементу програми, адресу якого може зберігати даний покажчик. Це може бути будь який допустимий у мові тип, простий або складений

Ім'я покажчика – це ідентифікатор написаний за правилами запису імен у мові С.

* – це ознака того, що наступне ім'я є покажчиком.

Приклад оголошення покажчика з ім'ям px, на дані типу int наведено нижче:

```
int  *px ;
```

Слід мати на увазі, що символ * відноситься не до типу, а до імені покажчика, тому в разі оголошення декількох покажчиків символ * треба ставити перед кожним іменем. Нижче наведено приклад оголошення двох покажчиків pw, pz одночасно із оголошенням простої змінної pv:

```
int  *pw, *pz, pv ;
```

Під час оголошення покажчика його можна ініціалізувати константою цілого типу, але цю константу слід явно привести до типу покажчика. У прикладі наведеному нижче покажчику pd на дані типу double присвоюється значення 1024 (нагадаємо, що значення покажчика – це адреса даних):

```
double  *pd = (double*)1024;
```

Зверніть увагу, у конструкції, що використовується для приведення до типу покажчика символ * відноситься вже до назви типу даних.

Покажчик можна проініціалізувати і адресою деякої, вже оголошеної змінної. Для отримання адреси змінної використовується операція визначення адреси - &. Нижче наведено приклад такої ініціалізації:

```
double  z, *pz = &z;
```

У цьому прикладі оголошені проста змінна z, і покажчик pz, який проініціалізований адресою змінної z. Таким чином вказівник pz містить адресу змінної z.

Якщо покажчик не проініціалізовано, то його значенням буде «сміття», тобто будь яка адреса. Для того, щоб не мати справи із «сміттям», у тих випадках, коли невідомо, як ініціалізувати покажчик, йому присвоюють значення NULL. Вважається, що покажчик, значенням якого є NULL, ні на що не посилається і його називають порожнім покажчиком. Нижче наведено приклад такої ініціалізації:

```
int  *pw = NULL ;
```

11.1.2 *Звернення до даних через покажчики*

Якщо відома адреса якихось даних, то мабуть можна отримати і значення цих даних. Операцію отримання даних через їх адресу називають операцією розадресації і позначають знову ж таки символом *, що ставиться перед ім’ям покажчика. Звертання до даних через покажчик можна використовувати всюди, де можна звертатися до даних через ім’я змінної.

Нехай оголошено зміну z і проініціалізовано покажчик на неї pz:

```
double  z, *pz = &z;
```

У цьому випадку вирази

```
z = 12.37;  z  += 2.5;  z *= 2;
```

можна записати і так:

```
*pz = 12.37;  *pz  += 2.5;  *pz *= 2;
```

Як бачимо з цього прикладу, вирази &z та pz позначають адресу змінної z, а вирази z та *pz позначають поточне значення змінної z.

Працюючи з покажчиками у програмах на мові С слід пам’ятати, що ніякого контролю за значеннями покажчиків, що використовуються для

доступу до даних нема. Ви можете звернутися до «своїх даних» з будь якою адресою, але отримаєте невідомо що. Ще гірше, якщо ви зміните значення цих «своїх даних». Це може призвести до катастрофічних наслідків для вашої програми.

11.1.3 *Використання кваліфікатора const для покажчиків*

Покажчики можна оголошувати з кваліфікатором const. При цьому можливі три варіанти оголошення.

Перший варіант полягає у тому, що кваліфікатор const забороняє змінювати дані на які він посилається. Для реалізації цього варіанту слово const має бути записано перед типом покажчика:

```
const int z = 100;
const int *ptc = &z;
```

Зверніть увагу, ініціалізувати покажчик на константні дані можна тільки адресою константи.

Покажчики на константні дані часто використовуються в оголошеннях параметрів функцій.

Другий варіант використання кваліфікатора const передбачає незмінність самого покажчика. Для реалізації цього варіанту слово const має бути записано перед іменем покажчика:

```
int  *const cpt = &z;
```

Оголошений таким чином покажчик cpt не може змінювати свого значення, а змінна z на яку він посилається змінюватися може. Такі покажчики називають константними.

Третій варіант використання кваліфікатора const передбачає незмінність як самого покажчика, так і даних, на які він посилається. Для реалізації цього варіанту слово const має бути записано двічі, перед типом та перед іменем покажчика:

```
const int z = 100;
const int *const ptc = &z;
```

11.1.4 *Адресна арифметика*

Над покажчиками можливі такі операції:

- присвоєння;
- порівняння;
- збільшення/зменшення;
- віднімання.

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.

Для порівняння покажчиків можна використовувати звичайні операції порівняння. Найчастіше з цих операцій використовуються операції == та !=.

До значення покажчиків можна додавати (або віднімати) цілі числа. При цьому значення покажчика збільшується на величину числа, що додається, помножену на кількість байтів, що займає у пам'яті елемент даних відповідного типу. У результаті виконання наведеного нижче прикладу значення покажчика ptn буде на 20 більшим, ніж значення покажчика ptz. Тут мається на увазі що елемент даних типу int займає 4 байти.

```
int z = 100;
int  *ptn, *ptz = &z;
ptn = ptz +5 ;
```

Таким чином, вираз, у якому збільшується або зменшується значення покажчика, наприклад, на величину k, формує адресу елементу даних, розташованого на k елементів правіше, або лівіше у разі операції віднімання.

Для зміни значень покажчиків можна також застосовувати операції інкременту та декременту. Операція інкременту зміщує покажчик до наступного елементу, а декременту – до попереднього. Найчастіше такі операції використовуються під час роботи з масивами.

Операція віднімання, що виконується над двома покажчиками повертає кількість елементів базового, типу що може розміститися між адресами, на які вказують перший та другий операнди операції.

11.1.5 *Нетипізовані вказівники*

У мові С можна використовувати покажчики, які не пов'язані з якимось конкретним типом і сумісні покажчиками на будь які типи даних. Використання безтипових покажчиків дозволяє підвищувати ефективність створюваних програм. Безтипові покажчики оголошуються із ключовим словом void:

```
void *pz;
```

Хоча значення без типових покажчиків можна прямо присвоювати вказівникам на будь які типи даних, усе ж доцільно у таких присвоєннях використовувати явне приведення типу, бо це підвищує надійність програм.

Розглянемо приклад, який дозволяє отримати доступ до двох половин числа типу long (long удвічі більший за int).

```
ulong number = 0x12AB34DC;
uint  part1, part2;
void *p= &number;
part1=*(uint*)p;
part2=*((uint*)p+1)
```

У цьому прикладі ми розрізали число number типу long на числа part1та part2 типу int.

11.2 ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

В лабораторній роботі слід створити проект, передбачає реалізацію таких функцій у проекті:

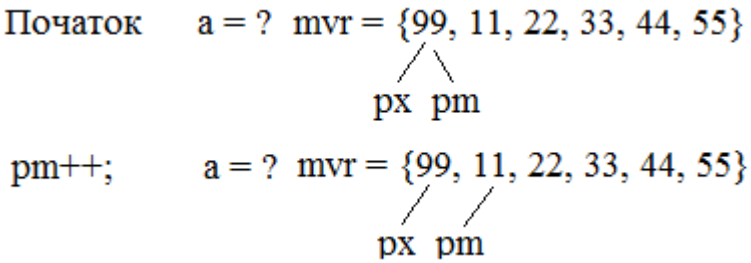
– виведення на консоль у вигляді таблиці значень чисел у десятковому та шістнадцятковому форматі, та їх адрес у оперативній пам'яті для типів int, long, float, double, char, bool;

– представити ті самі числа у вигляді окремих байтів та вивести їх на консоль;

– виконати без комп'ютера, а потім перевірити результат за допомогою комп'ютера приклад із підручника [4], стр.120:

```
//Дано:
   int mvr={99, 11, 22, 33, 44, 55};
   int *pm = mvr;
   int a, *px = pm;
//Треба знайти результат кожного з наступних операторів:
   pm++;
   a = *pm++;
   a = (*pm)++;
   a = *++pm;
   a = ++*pm;
   *px++ = *pm++;
```

Для зручності роботи результати виконання кожного шагу рекомендуємо представляти у звіті за такою схемою:



– виконати без комп'ютера, а потім перевірити результат за допомогою комп'ютера приклад 3 із підручника [4], стр.124:

```
//Яким буде x після виконання даної програми:
   int x, *vk;
```

```
vk=&x;
x=5;
*vk = x * 5;
x+=*vk;
```

– виконати без комп'ютера, а потім перевірити результат за допомогою комп'ютера приклад 8 із підручника [4], стр.124:

```
//Яким буде n після виконання даної програми:
double *px;
double * const py = (double*)200;
int n = 0;
px = py +50;
px += 10;
px = px >= (double*) 600 ? px - 10 : NULL ;
if (px !=NULL)
   n = px - py;
```

11.3 ВИМОГИ ДО ЗВІТУ

– Назва роботи.
– Мета роботи.
– Короткий опис основних понять, пов'язаних з покажчиками.
– Тексти виконаних завдань з проміжними та конечними результатами.
– Результати тестування проекту у вигляді копій консолі.
– Висновки.

11.4 КОНТРОЛЬНІ ПИТАННЯ

– Визначення поняття покажчик.
– Оголошення та ініціалізація покажчиків.
– Звернення до даних через покажчики.
– Використання кваліфікатора const для покажчиків.
– Адресна арифметика.
– Нетипізовані вказівники.
– Протестувати програму обробки покажчиків наданої викладачем.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программированя С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с

2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. –1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вильямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

12 ЛАБОРАТОРНА РОБОТА № 12. ВИКОРИСТАННЯ ПОКАЖЧИКІВ ДЛЯ ОБРОБКИ МАСИВІВ ТА СТРУКТУР

Мета роботи:

- Навчитися обробляти одновимірні масиви, використовуючи покажчики.
- Ознайомитися з особливостями обробки масивів символів з використанням покажчиків.
- Навчитися оперувати з двовимірними масивами за допомогою покажчиків.
- Створити програму для обробки різних типів масивів використовуючи покажчики.

12.1 КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Можливість використання покажчиків для звертання до елементів масивів пов'язана з тим, що ім'я масиву є константним покажчиком на перший елемент масиву. Таким чином, якщо маємо оголошення масиву:

```
int a[10];
```

то звертання *a дає той самий результат, що й a[0]. Звертання *(a+3) дасть той самий результат, що й a[3].

12.1.1 *Приклад формування одновимірного масиву з використанням покажчиків*

Нижче наведено приклад створення масиву для збереження чисел Фібоначчі, який розглядався у лабораторній роботі номер 7. Два перших числа Фібоначчі рівні 0 та 1, а кожне наступне дорівнює сумі двох попередніх. У програмі перші 2 елементи масиву заповнюються до циклу, решта - в циклі.

```
int a[10];
  *a = 0;
  *(a+1) = 1;
  for (i=2 ; i<10; i++)
     *(a+i) = *(a + i – 2) + *(a + i – 1);
```

У цьому прикладі доступ до елементів масиву здійснюється за допомогою покажчиків, але самі покажчики формуються за допомогою операцій адресної арифметики з використанням адреси масиву та індексу елементів i, що формується у циклі for.

Але частіше для обробки елементів масиву використовуються додаткові покажчики. За звичай один такий покажчик вказує на кінець масиву, а другий, поточний, формується у циклі for.

Нижче наведено такий варіант формування масив чисел Фібоначчі.

```
    int n=10, a[n],  *pCurrent, *pEnd;
    pEnd = a+n;
    pCurrent = a;
    *pCurrent++ = 0;
    *pCurrent++ = 1;
    for ( ; pCurrent <pEnd; pCurrent ++)
        * pCurrent  = *( pCurrent – 2) + *( pCurrent – 1);
```

12.1.2 *Робота з масивами символів через покажчики*

Так само як и у випадку числового масиву, ім'я символьного масиву є константним покажчиком на перший символ. Тому для обробки символьних рядків теж можна використовувати покажчики, так само, як і для масивів чисел. Обробка навіть дещо спрощується завдяки тому, що кожний рядок закінчується спеціальним символом '\0'.

У якості прикладу розглянемо функцію пошуку позиції, з якої комбінація символів sub знаходиться у рядку str:

```
int posInStr( char *str,  char *sub){
   char *p1, *p2, *p;
   for(p1=str; ; p1++){
      for(p=p1,p2=sub; *p != '\0' && *p2 != '\0' && *p == *p2 ; p++, p2++);
      if(*p2=='\0')return p1-str;
      if(*p=='\0')return -1;
   }
}
```

В якості параметрів до цієї функції передаються покажчики на рядок і комбінацію символів.

Зовнішній цикл for забезпечує послідовне переміщення по символах рядка.

Внутрішній цикл, який не має тіла, забезпечує порівняння символів комбінації із символами рядка, починаючи з поточного. Він має три умови завершення.

Якщо поточні символи комбінації і рядка не співпадають, відбувається перехід до зовнішнього циклу.

Якщо під час цього порівняння знайдено кінець комбінації то це означає, що комбінація входить до складу рядка, і функція повертає відстань між поточним символом рядка і його початком.

Якщо знайдено кінець рядка, то повертається ознака того, що комбінація не входить до рядка.

12.1.3 *Стандартні функції для роботи з рядками символів*

Стандартні бібліотеки мови С надають користувачеві можливість використовувати ряд функцій для опрацювання символьних рядків, а також функцій перетворень рядків символів у числа та зворотних перетворень. Ці функції докладно описані у підручнику [4], стр.152-154. Ознайомтесь із цими функціями і перевірте їх роботу. Майте на увазі, у підручнику є помилка, яку не важко помітити.

У звіті наведіть результати тестування деяких з цих функцій (не менше 2 з кожної групи.)

12.1.4 *Масиви покажчиків на рядки символів*

У практиці програмування зустрічаються задачі в яких доводиться працювати з масивами рядків символів. Прикладом такого масиву може бути список студентів групи. Особливість таких масивів полягає у тому, що майже кожний елемент має свою довжину. Якщо оголошувати такий масив як двовимірний (матриця символів), то ширину такої матриці доведеться вибирати виходячи з найдовшого прізвища. Це призведе до нераціонального використання пам'яті.

Значно краще створити одновимірний масив покажчиків на прізвища і з ним працювати. Нижче наведено приклад, у якому створюється масив покажчиків на рядки. Після ініціалізації масив упорядковується і виводиться на консоль.

```
//Створення масиву покажчиків на рядки
const char *arrPtrToFio[]={"Жванецький М.В.","Вовк А.П.","Карась П.С."};
//Сортування покажчиків методом вибору
for(int i=0; i<2;i++){
   for(int j=1;j<3;j++){
      //Порівняння прізвищ
      if(strcmp(arrPtrToFio[i], arrPtrToFio[j])>0){
         // Обмін покажчиків у масиві
         const char *p=arrPtrToFio[i];
         arrPtrToFio[i]=arrPtrToFio[j];
         arrPtrToFio[j]=p;
      }
   }
}
//Виведення на консоль
```

```
for(int i=0;i<3;i++){
    cout<<arrPtrToFio[i]<<endl;
}
```

Зверніть увагу, під час сортування обмін відбувається тільки між покажчиками, а рядки залишаються на місцях.

Ілюстрації до прикладу можна знайти у підручнику [4], стр.159-160.

12.1.5 *Покажчики на структури*

Змінні, яким відповідають структури, розглядаються у мові С як звичайні змінні. До цих змінних можна застосовувати операцію &, яка поверне адресу першого байту ділянки оперативної пам'яті, що займає відповідна структура.

Можна оголошувати покажчик на структуру, використовуючи назву шаблону структури в якості базового типу покажчика.

Використовуючи операції розадресації слід для структур слід пам'ятати, що операція «крапка» має вищий пріоритет, ніж операція «зірочка». Тому можна використовувати такий вираз для отримання першого символу рядка name, що є полем структури: *bestStud.name.

До полів структури можна звертатися і використовуючи покажчик на цю структуру. Але у цьому разі замість операції «крапка» слід використовувати операцію ->. Слід зазначити, що Qt автоматично змінює операцію «крапка» на операцію -> у разі помилки з вибором операції.

Покажчики на структури можна використовувати і в якості полів структури, у тому числі і у структурах того самого типу. Такі поля є невід'ємною частиною динамічних інформаційних структур – списків та дерев.

12.2 ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

В лабораторній роботі слід створити проект, у якому ви маєте відповідно до вимог варіантів на обробку масивів та їх сортування з попередніх лабораторних робіт написати, **використовуючи покажчики**, та протестувати такі функції:

- визначення числової характеристики для створеного масиву;
- формування нового масиву на основі створеного;
- сортування масиву за ускладненим правилом.

12.3 ВИМОГИ ДО ЗВІТУ

- Назва роботи.
- Мета роботи.
- Короткий опис особливостей роботи з масивами через покажчики.
- Огляд функцій для роботи із рядками символів.
- Тексти функцій для індивідуальних завдань з коментарями.
- Результати тестування завдань у вигляді копій консолі.
- Висновки.

12.4 КОНТРОЛЬНІ ПИТАННЯ

- Використання покажчиків у роботі з масивами.
- Використання покажчиків у роботі із рядками символів.
- Масиви покажчиків на рядки символів та переваги їх використання.
- Використання покажчиків у роботі із структурами.
- Написати функцію для обробки масиву чисел з використанням покажчиків за вказівкою викладача.
- Написати функцію для обробки рядка символів з використанням покажчиків за вказівкою викладача.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Берн Страуструп. Язык программированя С++. Второе дополненное издание. – М: Бином-Пресс, 2008. – 369 с
2. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.:ООО «ДиаСофтЮП», 2003. –1104 с.
3. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вильямс, 2004. 783 с.
4. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.