МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Чернігівська політехніка»

ОСНОВИ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт та самостійної роботи
для здобувачів вищої освіти
за освітніми програмами “Комп’ютерна інженерія” та
«Інженерія програмного забезпечення»
(освітній ступінь бакалавр)

Обговорено і рекомендовано
на засіданні кафедри
інформаційних і комп'ютерних
систем
Протокол № 1 від 25.01.24

Чернігів 2024

Основи програмування. Методичні вказівки до виконання лабораторних робіт та самостійної роботи для здобувачів вищої освіти за освітньою програмою „Комп'ютерна інженерія" та «Інженерія програмного забезпечення». /Укл.: Бивойно П.Г., Бивойно Т.П., Якименко І.В. – Чернігів, 202. – 184 с.

Укладачі: Бивойно Павло Георгійович, канд. техн. наук, доцент
Бивойно Тарас Павлович, ст. викладач
Якименко Ірина Вікторівна, асистент

Відповідальний за випуск: Базилевич В.М., зав. кафедрою інформаційних и комп'ютерних систем, канд.економ.наук.

Рецензент: Пріла О.А., канд. техн. наук, доцент кафедри інформаційних та комп'ютерних систем НУ «Чернігівська політехніка»

ЗМІСТ

Зміст 3
Вступ 13
1 Лабораторна робота 1. Знайомство з інтегрованим середовищем розробки Code::Blocks 14
1.1 Мета роботи 14
1.2 Завдання на роботу 14
1.3 Теоретична частина 15
1.3.1 Короткі відомості про IDE Code::Blocks 15
1.3.2 Проєкт Code::Blocks 15
1.3.3 Програма на мові C\C++ 15
1.3.3.1 Функція main 16
1.3.3.2 Коментарі 17
1.3.3.3 Файл з декількома функціями 17
1.3.4 Правила запису констант в мові C/C++ 18
1.3.4.1 Константи цілих чисел 18
1.3.4.2 Переведення десяткових цілих чисел в інші системи 18
1.3.4.3 Константи дійсних чисел 19
1.3.4.4 Символьні константи 20
1.3.4.5 Рядки символів 20
1.3.5 Виведення та введення даних 20
1.3.5.1 Виведення інформації на консоль 20
1.3.5.2 Підготовка для введення даних з консолі 21
1.3.5.3 Засоби для введення даних з консолі 21
1.3.5.4 Проблеми з введенням даних 22
1.3.6 Використання математичних функцій 22
1.4 Рекомендації до виконання роботи 23
1.4.1 Підготовка до роботи 23
1.4.1.1 Встановлення Code::Blocks 23
1.4.2 Створення найпростішого проєкту 23
1.4.3 Деякі налаштування середовища розробки 25
1.4.3.1 Редагування і налаштування української мови 25
1.4.3.2 Форматування тексту програми 26
1.4.3.3 Пошук заміна в тексті 26
1.4.3.4 Деякі налаштування редактора та меню 26
1.4.3.5 Налаштування панелі інструментів 27
1.4.3.6 Налаштування консолі 27
1.4.4 Шаблон для лабораторних проєктів 27
1.4.4.1 Створення шаблону 28

1.4.4.2 Збереження шаблону 28
1.4.4.3 Використання шаблону для створення проєкту 28
1.4.5 Реалізація тестування форм представлення чисел 29
1.4.6 Введення даних та простими розрахунками 30
1.4.7 Експерименти з варіантами введення рядків символів..... 30
1.4.8 Реалізація завдання з підручника 30
1.4.9 Експерименти з функцією з підручника..... 31
1.5 Завдання для самостійної роботи 31
1.6 Вимоги до звіту 31
1.6.1 Основні вимоги до оформлення 31
1.6.2 Вміст звіту..... 32
1.7 Контрольні питання 33
2 Лабораторна робота 2. Типи даних та розрахунки за формулами..... 34
2.1 Мета роботи..... 34
2.2 Завдання на лабораторну роботу..... 34
2.3 Теоретична частина..... 34
2.3.1 Змінні та константи..... 34
2.3.2 Типи даних..... 34
2.3.3 Арифметичні типи даних 35
2.3.3.1 Основні арифметичні типи 35
2.3.3.2 Модифіковані арифметичні типи 36
2.3.3.3 Граничні значення даних цілочислових типів даних..... 36
2.3.3.4 Типи цілочислових констант 37
2.3.3.5 Типи констант дійсних типів 37
2.3.4 Описи змінних..... 37
2.3.4.1 Звичайні змінні..... 37
2.3.4.2 Макроконстанти..... 38
2.3.5 Операція розміру sizeof 38
2.3.6 Арифметичні операції 39
2.3.7 Операції присвоєння..... 39
2.3.7.1 Проста операція присвоєння..... 39
2.3.7.2 Комбіновані присвоєння 40
2.3.7.3 Унарні присвоєння..... 40
2.3.8 Вирази 41
2.3.9 Пріоритети операцій у C++..... 41
2.3.10 Узгодження типів..... 42
2.3.10.1 Арифметичні перетворення 42
2.3.10.2 Перетворення типів в операціях присвоєння..... 43
2.3.10.3 Явне перетворення типів..... 43

2.3.11 Бібліотека математичних функцій cmath 43
2.3.12 Форматне виведення даних через функцію printf() 44
2.4 Рекомендації до виконання роботи 46
2.4.1 Програма реалізації операцій з даними різних типів 46
2.4.2 Реалізація розрахунків за формулами 47
2.4.3 Форматне виведення розміру основних типів даних та їх граничних значень 48
2.5 Завдання для самостійної роботи 48
2.6 Вміст звіту 48
2.7 Контрольні питання 49
3 Лабораторна робота 3. Функції 50
3.1 Мета роботи 50
3.2 Завдання на лабораторну роботу 50
3.3 Теоретична частина 51
3.3.1 Правила написання функцій 51
3.3.2 Виклик функції 52
3.3.3 Прототип функції 53
3.3.4 Прототипи бібліотечних функцій 54
3.3.5 Способи передачі параметрів у функції 54
3.3.5.1 Передача параметрів за значенням 54
3.3.5.2 Передача параметрів через посилання 55
3.3.6 Області оголошення та доступу до імен 56
3.3.6.1 Глобальні та локальні змінні 56
3.3.6.2 Глобальна чи локальна змінна? 56
3.3.6.3 Специфікатор static 57
3.4 Рекомендації до до виконання роботи 57
3.4.1 Аналіз завдання та контрольний розрахунок 58
3.4.1.1 Аналіз 58
3.4.1.2 Контрольний розрахунок 58
3.4.2 Створення проєкту 58
3.4.2.1 Прототипи функцій 58
3.4.2.2 Часткова реалізація функції main() 59
3.4.2.3 Реалізація функції з поверненням результату через return 59
3.4.2.4 Остаточна реалізація функції main() для даного завдання 59
3.4.2.5 Реалізація функції з поверненням результату через посилання 60
3.4.3 Робота з програмою в режимі налагодження 60

3.4.3.1 Покрокове виконання програми. 61
3.4.3.2 Використання точок переривання 61
3.4.4 Дослідження передачі параметрів за значенням 61
3.4.5 Дослідження передачі параметрів за посиланням 62
3.5 Завдання для самостійної роботи 62
3.5.1 Рекомендації до виконання завдання 62
3.6 Вміст звіту 64
3.7 Контрольні питання 64
4 Лабораторна робота 4. Логічний тип даних і розгалуження у програмах 65
4.1 Мета роботи 65
4.2 Завдання на лабораторну роботу 65
4.3 Теоретична частина 65
4.3.1 Логічний тип даних 65
4.3.1.1 Операції над даними логічного типу 66
4.3.1.2 Логічні вирази 66
4.3.2 Алгоритми з розгалуженнями 67
4.3.3 Програмування розгалужень 69
4.3.3.1 Оператор розгалуження if…else 69
4.3.3.2 Умовна операція 71
4.3.3.3 Оператор вибору switch 71
4.3.4 Оператор переходу goto 73
4.4 Рекомендації до виконання роботи 73
4.4.1 Створення проєкту 73
4.4.1.1 Прототипи функцій 74
4.4.1.2 Часткова реалізація функції main() 74
4.5 Завдання для самостійної роботи 76
4.6 Вимоги до звіту 76
4.7 Контрольні питання 77
5 Лабораторна робота 5. Використання циклів while та do…while 78
5.1 Мета роботи 78
5.2 Завдання на роботу 78
5.3 Теоретична частина 79
5.3.1 Циклічні алгоритми 79
5.3.2 Оператор while 80
5.3.3 Оператор do…while 82
5.3.4 Переривання циклу 83
5.3.5 Цикл while з умовою всередині тіла циклу 83

5.3.6 Ітераційні алгоритми 84
5.3.7 Алгоритми обчислення сум нескінченних рядів 84
5.4 Рекомендації до виконання роботи 85
5.4.1 Реалізація ітераційного алгоритму 85
5.4.1.1 Формулювання завдання 85
5.4.1.2 Аналіз завдання 85
5.4.1.3 Реалізація функції пошуку кореня 86
5.4.1.4 Реалізація функції main для тестування функції пошуку кореня 86
5.4.2 Реалізація пошуку суми нескінченного ряду 87
5.4.2.1 Формулювання завдання 87
5.4.2.2 Аналіз завдання 87
5.4.2.3 Реалізація функція обчислення суми нескінченного ряду 88
5.4.2.4 Друга частина функції main() 88
5.1 Вимоги до звіту 89
5.2 Контрольні питання 89
6 Лабораторна робота 6. Обробка даних за допомогою циклу for 90
6.1 Мета роботи 90
6.2 Завдання на роботу 90
6.3 Теоретична частина 92
6.3.1 Оператор циклу for 92
6.3.1.1 Алгоритм виконання циклу for 92
6.3.1.2 Обчислення числа Фібоначчі у циклі for 93
6.3.1.3 Обчислення факторіалу у циклі for 93
6.3.2 Особливості використання циклу for 94
6.3.3 Випадкові числа 95
6.4 Рекомендації до виконання роботи 96
6.4.1 Табулювання функцій 96
6.4.1.1 Завдання на табулювання функції 96
6.4.1.2 Аналіз завдання 96
6.4.1.3 Реалізація функції для розрахунків за формулою 97
6.4.1.4 Реалізація функції введення і контролю даних 97
6.4.1.5 Реалізація функції виведення таблиці значень на консоль 97
6.4.1.6 Реалізація першого завдання в функції main() 98
6.4.2 Обробка послідовностей цілих чисел 98
6.4.2.1 Формулювання завдання. 98
6.4.2.2 Аналіз завдання 99

6.4.2.3 Розробка функції тестування числа 99
6.4.2.4 Реалізація функції обробки послідовності чисел 99
6.4.2.5 Реалізація другої частини у функції main() 100
6.4.3 Обробка послідовності випадкових чисел 100
6.4.3.1 Формулювання завдання. 100
6.4.3.2 Розробка функції, що обчислює задану характеристику 100
6.4.3.3 Реалізація останньої частини функції main 101
6.5 Вимоги до звіту 101
6.6 Контрольні питання 101
7 Лабораторна робота 7. Основи роботи з вказівниками 102
7.1 Мета роботи 102
7.2 Завдання на роботу 102
7.3 Теоретична частина 102
7.3.1 Оголошення та ініціалізація вказівників 103
7.3.2 Звернення до даних через вказівники 104
7.3.3 Використання кваліфікатора const для вказівників 104
7.3.4 Адресна арифметика 105
7.3.5 Нетипізовані вказівники 106
7.3.6 Вказівники, як параметри функцій 106
7.3.6.1 Передача вказівників за значенням 106
7.3.6.2 Передача вказівників за посиланням 107
7.3.6.3 Вказівник як результат функції 107
7.3.7 Вказівники на функції 108
7.3.7.1 Тип функції 108
7.3.7.2 Використання вказівників на функції 108
7.3.8 Операції з пам’яттю 109
7.3.8.1 Отримання доступу до ділянки динамічної пам’яті 110
7.3.8.2 Звільнення пам’яті 111
7.3.8.3 Стандартні функції для роботи з ділянками оперативної пам'яті 111
7.4 Рекомендації до виконання роботи 112
7.4.1 Реалізація завдання 1 112
7.4.1.1 Аналіз завдання 112
7.4.1.2 Реалізація функції виведення на консоль чисел з ділянки пам’яті 112
7.4.1.3 Реалізація частини функції main() для першого завдання 113
7.4.2 Реалізація завдання 2 113

7.4.2.1 Формулювання завдання. 113
7.4.2.2 Аналіз завдання 113
7.4.2.3 Реалізація функція тестування вибірки випадкових чисел 114
7.4.2.4 Друга частина функції main() 114
7.4.3 Реалізація завдання 3 115
7.4.3.1 Формулювання завдання. 115
7.4.3.2 Аналіз завдання 115
7.4.3.3 Функція обчислення характеристики для вибірки випадкових чисел 115
7.4.3.4 Третя частина функції main() 116
7.4.1 Реалізація завдання 4 116
7.5 Вимоги до звіту 117
7.6 Контрольні питання 117
8 Лабораторна робота 8. Робота з одновимірними масивами 118
8.1 Мета роботи 118
8.2 Завдання на роботу 118
8.3 Теоретична частина 119
8.3.1 Оголошення масиву та звернення до його елементів 120
8.3.2 Приклад використання одновимірного масиву 121
8.3.3 Одновимірні масиви як параметри функцій 121
8.3.4 Функції обробки масивів чисел 122
8.3.4.1 Функція введення масиву з консолі по елементам 123
8.3.4.2 Функція формування випадкового масиву 123
8.3.4.3 Функція виведення масиву на консоль 123
8.3.4.4 Функція пошуку індексу елемента масиву за його значенням 124
8.3.4.5 Пошук максимального і мінімального елементів масиву та їх індексів 124
8.3.4.6 Функція видалення елемента з масиву 125
8.3.4.7 Функція перевороту масиву 125
8.3.5 Функції формування нового масиву на основі існуючого 126
8.3.5.1 Функція формування масиву накопичених значень елементів. 126
8.3.5.2 Функція, що створює новий масив з невідомим наперед розміром 126
8.4 Рекомендації до виконання роботи 127
8.4.1 Реалізація завдання 1 127
8.4.1.1 Аналіз завдання 127

8.4.1.2 Реалізація частини функції main() для першого завдання 128
8.4.2 Реалізація завдання 2 128
8.4.2.1 Аналіз завдання 128
8.4.2.2 Реалізація функції підрахунку добутку чисел масиву 128
8.4.2.3 Реалізація частини функції main() для другого завдання 129
8.4.3 Реалізація завдання 3 129
8.4.3.1 Аналіз завдання 129
8.4.3.2 Реалізація частини функції main() для третього завдання 129
8.5 Вимоги до звіту 129
8.6 Контрольні питання 130
9 Лабораторна робота 9. Впорядковані масиви 131
9.1 Мета роботи 131
9.2 Завдання на лабораторну роботу 131
9.2.1 Вимоги до масивів 132
9.2.2 Вимоги до переліку функцій 132
9.2.3 Вимоги до функції main() 132
9.3 Теоретична частина 133
9.3.1 Використання вказівника на функцію порівняння елементів у функціях сортування масивsd 133
9.3.2 Алгоритми сортування масивів 134
9.3.2.1 Методо вибору 134
9.3.2.2 Сортування масиву методом обміну 135
9.3.2.3 Сортування масиву за методом вставки 137
9.3.3 Обробка впорядкованих масивів 139
9.3.3.1 Злиття впорядкованих масивів 139
9.3.3.2 Пошук позиції елемента у впорядкованому масиві 140
9.3.3.3 Вставка елемента у впорядкований масив 140
9.3.3.4 Видалення елемента з упорядкованого масиву 142
9.4 Рекомендації до виконання роботи 143
9.5 Вимоги до звіту 145
9.6 Контрольні питання 145
10 Лабораторна робота 10. Рядки символів 146
10.1 Мета роботи 146
10.2 Завдання на лабораторну роботу 146
10.3 Теоретична частина 147
10.3.1 Оголошення рядка 147

10.3.2 Особливості зчитування рядків символів 148
10.3.3 Кодування символів у рядках 149
10.3.4 Реалізація функцій для обробка рядків символів 150
10.3.4.1 Функція, що обчислює кількість символів у рядку 150
10.3.4.2 Функція копіювання частини рядка 150
10.3.4.3 Функція знаходження позиції групи символів у рядку 151
10.3.4.4 Функція, що вилучає із рядка зайві пробіли 151
10.3.4.5 Перетворення цілого числа в рядок символів 152
10.3.4.6 Перетворення рядка символів у ціле число 153
10.3.5 Стандартні функції для роботи з рядків символів 153
10.3.5.1 Стандартні функції, що оперують з окремими символами. 153
10.3.5.2 Стандартні функції для прямих та зворотних перетворень між рядками символів та числами. 154
10.3.5.3 Стандартні функції для роботи з рядками символів. .. 154
10.4 Рекомендації до виконання роботи 155
10.4.1 Завдання 1 155
10.4.1.1 Аналіз завдання 156
10.4.1.2 Приклад виведення результатів тестування символів 156
10.4.2 Завдання 2 156
10.4.2.1 Аналіз завдання 156
10.4.2.2 Реалізація другої частини функції main() 157
10.4.3 Завдання 3 157
10.4.3.1 Аналіз завдання 157
10.4.3.2 Реалізація функції 157
10.4.3.3 Реалізація третьої частини функції main() 158
10.5 Вимоги до звіту 158
10.6 Контрольні питання 158
11 Лабораторна робота 11. Структури 159
11.1 Мета роботи 159
11.2 Завдання на лабораторну роботу 159
11.3 Теоретична частина 159
11.3.1 Оголошення шаблону та ініціалізація структур 159
11.3.2 Масиви структур 161
11.3.3 Введення-виведення структур 162
11.3.4 Сортування масивів структур 162
11.4 Рекомендації до виконання роботи 162
11.4.1 Створення допоміжних функцій 163

11.4.2 Визначення глобальних змінних та констант 164
11.4.3 Початкова ініціалізація та демонстрація масиву 164
11.4.3.1 Ініціалізація масиву 165
11.4.3.2 Функція відображення масиву на консолі 165
11.4.4 Інтерфейс користувача для проєкту 165
11.4.5 Створення елементів меню 166
11.4.5.1 Функція активізації меню. 167
11.4.6 Додавання нових даних до масиву структур 167
11.4.7 Видалення запису з масиву 169
11.4.8 Функція сортування масиву структур 169
11.4.9 Реалізація сортування за ім’ям студента 170
11.4.10 Реалізація сортування за результатами навчання 170
11.4.11 Вибірка студентів по граничному балу 171
11.4.12 Інформація про студентів що мають заборгованості 171
11.5 Вимоги до звіту 172
11.6 Контрольні питання 172
12 Лабораторна робота 12. Файли 173
12.1 Мета роботи 173
12.2 Завдання на лабораторну роботу 173
12.3 Теоретична частина 173
12.3.1 Відкриття та закриття файлу 173
12.3.2 Обробка текстових файлів 174
12.3.3 Обмін блоками даних з файлом 176
12.3.3.1 Збереження масиву у файлі 176
12.3.3.2 Відновлення масиву з файлу 176
12.3.4 Робота із записами файлу 177
12.3.5 Робота з файлом в режимі прямого доступу 178
12.4 Рекомендації до виконання роботи 180
12.4.1 Розширення переліку глобальних змінних 180
12.4.2 Створення меню для роботи з файлом 180
12.4.3 Створення файлу протоколу 180
12.4.4 Реалізація функції showProtocol 181
12.4.5 Функції збереження масиву у файлі 181
12.4.6 Функція виведення на консоль інформації із файлу 181
12.4.7 Функція відновлення масиву із файлу 182
12.4.8 Функція додавання запису до файлу 182
12.5 Вимоги до звіту 183
12.6 Контрольні питання 183

ВСТУП

Лабораторні роботи є сполучною ланкою між лекціями та самостійною роботою студентів. В процесі виконання лабораторних робіт експериментально перевіряються ключові питання курсу програмування, набуваються практичні навички побудови та налагодження програм, перевіряється ступінь засвоєння основних положень предмету. Під час лабораторних занять студенти знайомляться з типовими рішеннями деяких задач програмування.

Лабораторні роботи виконуються на персональних комп'ютерах в інтегрованому середовищі розробки програм Code::Blocks [1]. Передбачається, що студенти знайомі з основами роботи на персональному комп'ютері. Якщо таких навичок немає, то студент повинен придбати їх під час самостійної роботи в лабораторії. 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.

По кожній роботі студент повинен оформити звіт. Звіти оформляються в електронному вигляді за допомогою текстового редактора Word на сторінках формату А4, у відповідності з вимогами стандарте на оформлення технічної документації, зокрема, розділу 7 стандарту ДСТУ3008:2015 [2].

Окрім завдань, що вирішуються під час лабораторної роботи студент має виконати завдання для самостійної роботи. Результати виконання роботи у вигляді файлу з кодом програми і звіт по роботі студент завантажує на сайт дистанційного навчання, дотримуючись визначених термінів.

Якщо у викладача є питання до результатів роботи, студент має «захистити» свою роботу. Захист полягає у відповідях на питання по темі лабораторної роботи і внесенні деяких змін у розроблений проект, в присутності викладача.

З деяких тем студенти проходять тести, або пишуть контрольні роботи.

За лабораторну роботу студент може отримати до 100 балів, з урахуванням своєчасності та якості виконання всіх складових роботи. Складовими є: проект, звіт, відповіді на контрольні питання, результати тестів та контрольних. 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.

Цей варіант методичних вказівок є результатом переробки та доповнення методичних вказівок «Основи програмування» видання 2019 року.

1 ЛАБОРАТОРНА РОБОТА 1. ЗНАЙОМСТВО З ІНТЕГРОВАНИМ СЕРЕДОВИЩЕМ РОЗРОБКИ CODE::BLOCKS

1.1 Мета роботи

– Познайомитися з простою програмою на C/C++.
– Отримати первинні навички роботи з IDE Code::Blocks.
– Створити шаблон для майбутніх проєктів.
– Створити простий проект.

1.2 Завдання на роботу

– Встановити на комп'ютері середовище розробки Code::Blocks і налаштувати його відповідно до рекомендацій.
– Познайомитися з поняттям проєкт в середовищі Code::Blocks.
– Познайомитися з програмою «Hello World» на C/C++.
– Створити і зберегти в архіві шаблон для лабораторних проєктів.
– Створити на основі шаблону проєкт для лабораторної роботи.
– Додати до проєкту код для перевірки правильності переведення цілого додатного десяткового числа у двійкову, вісімкову і шістнадцяткову форми представлення. Використовувати число, яке є сумою року народження студента і дня народження
– Додати до програми фрагмент з введенням тексту і двох чисел, обчислення і виведення результату операції над цими числами, а також виведення результату обчислення заданої функції для суми чисел. Функцію і операцію вибирати з таблиці 1.1 відповідно до номера у списку групи. Якщо номер більше 10, віднімайте 10. Під час тестування програми в якості рядка символів треба вводити своє ім'я та прізвище, а для розрахунків вводити числа, що дорівнюють року і дню дати народження.

Таблиця 1.1 – Завдання на введення і обробку даних

Номер	1	2	3	4	5	6	7	8	9	10
Функція	sqrt	exp	log	log10	sin	cos	tan	sinh	cosh	tanh
Операція	+	-	/	*	%	+	-	/	*	%

– Проаналізувати різні варіанти введення рядків символів.
– Додати до проекту функцію і код звернення до неї відповідно до завдання 9, розділу 2 з підручника [3], стр.80. Протестувати створену програму.
– Провести експерименти з програмою, як рекомендує підручник у пункті 10, і написати у звіті висновки, яким чином впливали на роботу програми внесені зміни.

1.3 Теоретична частина

1.3.1 Короткі відомості про IDE Code::Blocks

На сьогодні відомо багато інструментів для розробки програм на мові C/C++. Одним із них є Code::Blocks. Це безкоштовне міжплатформне інтегроване середовище розробки (IDE) програм на мові програмування C/. Більше інформації про IDE Code::Blocks можна знайти за посиланням [1].

1.3.2 Проєкт Code::Blocks

В середовищі Code::Blocks, так само, як і в більшості інших середовищ розробки, програма створюється у вигляді проєкту.

Проєкт – це папка, яка містить файли пов'язані з програмою, що розробляється в межах проєкту. Тобто проєкт – це дерево каталогів з файлами.

Code::Blocks дозволяє працювати з декількома проектами. Якщо проєктів багато, то їх можна розподілити між робочими областями. Робоча область – це абстрактне поняття, з яким пов'язано декілька файлів, що знаходяться в системній папці Users\...\AppData\...CodeBlock. і містять посилання на проєкти робочої області. За замовчуванням користувач працює з робочою область Workspace.

1.3.3 Програма на мові C\C++

Програма на мові C\C++ являє собою сукупність функцій, які написані на мові C/C++ і розташовуються у файлах з розширенням .cpp.

Окрім функцій у файлах .cpp можна побачити інші складові – директиви, оголошення, тощо.

На рисунку 1.1 показано, як виглядає проста програма в Code::Block, яка створюється автоматично разом із створенням нового проєкту.

Як свідчить ліва панель, в робочій області Workspace знаходиться проєкт MyFirstProject, а в ньому файл main.cpp, який було створено автоматично.

У першому рядку файлу знаходиться директива препроцесора include, яка підключає до програми бібліотеку iostream, що забезпечує потокове введення-виведення інформації у програмі.

У рядку з номером 3 оголошується стандартний простір імен std.

У рядках з п'ятого по дев'ятий знаходиться функція main(). Таку функцію обов'язково повинна мати у своєму складі кожна програма на C/C++.

Це головна функція програми з якої починається виконання кожної програми. Функція main викликає інші функції, які, у свою чергу, можуть звертатися до потрібних їм функцій, і таким чином вирішується задача, для якої створено програму.

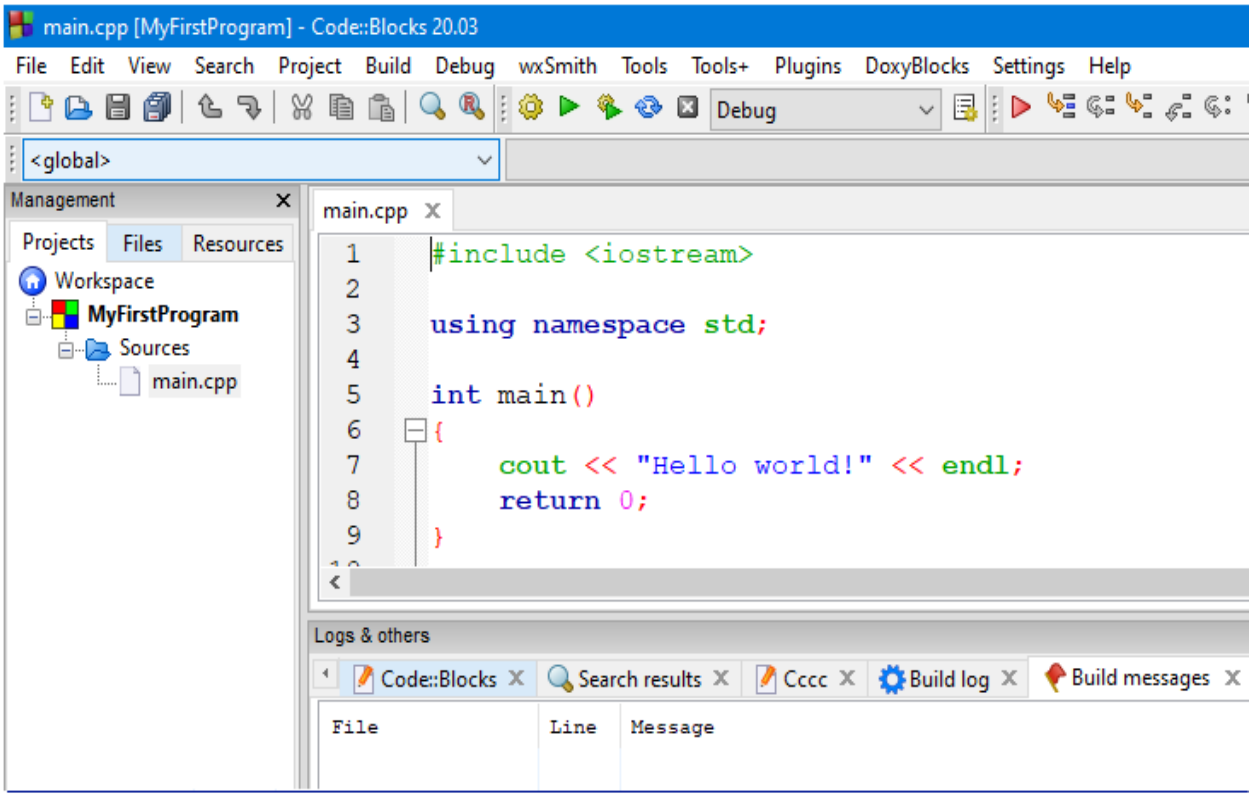


Рисунок 1.1 – Робоче вікно Code::Blocks

Кожна програма на C/C++ обов'язково повинна мати у своєму складі функцію з назвою main(). Це головна функція програми з якої починається виконання кожної програми. Функція main викликає інші функції, які, у свою чергу, можуть звертатися до потрібних їм функцій, і таким чином вирішується задача, для якої створено програму.

1.3.3.1 Функція main

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

Структуру функції поясняє рисунок 1.2.

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.

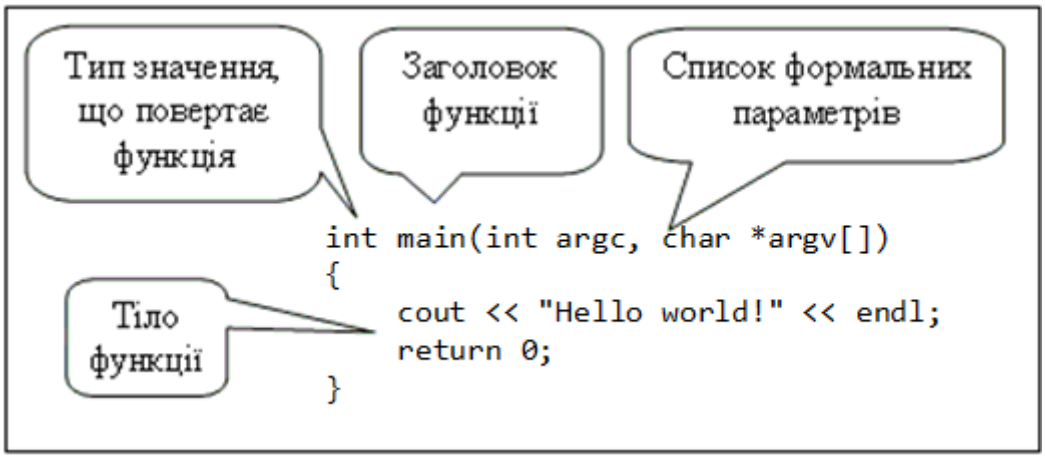


Рисунок 1.2 - Структура функції main()

У найпростішому випадку функція main() може мати і спрощений вигляд, лістинг 1.1.

Лістинг 1.1 – Функція main із спрощеним заголовком

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

Саме такий вигляд має функція, яка генерується автоматично під час створення нового проєкту.

У прикладах, що будуть наводитися далі, ми будемо використовувати саме такий, спрощений вигляд функції main().

1.3.3.2 Коментарі

Коментарі у мовах програмування використовуються для пояснень коду. І це дуже важливо! 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 //. Весь текст після цієї позначки і до кінця рядка є коментарем. Існують ще блочні коментарі мови C, які утворюються парами символів /* та */, але про них не варто було навіть згадувати.

1.3.3.3 Файл з декількома функціями

Наприкінці розділу 2 підручника [3] на стр.30 у пункті 9 завдань для самоконтролю наведено програму з двома функціями.

Суть завдання у пункті 10 полягає в тім, щоб студенти зрозуміли, навіщо потрібен прототип функції.

Якщо його закоментувати, то компілятор повідомить, що йому нічого не відомо про другу функцію, коли він опрацьовує першу.

1.3.4 Правила запису констант в мові C/C++

1.3.4.1 Константи цілих чисел

Цілі числа в програмах в мові C/C++ можна записувати не тільки у звичайній десятковій формі, але й у двійковій, вісімковій та шістнадцятковій формах.

Десяткова константа являє собою послідовність десяткових цифр, перед якою може бути знак + або -. Наприклад, 123, -234, +456. Нагадаємо, що десяткове число $435 = 4*10^2 + 3*10^1 + 5*10^0 = 400 + 30 + 5$.

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 ($1*8^2 + 2*8^1 + 5*8^0 = 1*64 + 2*8 + 5*1$) . Вісімкові константи раніше використовувалися для скорочення запису двійкових констант, бо кожну тріаду двійкових цифр можна замінити вісімковою цифрою. Нулю відповідає тріада 000, сімка виглядає як 111. Тобто число 0125 у двійковому коді виглядає як 1 010 101(пробіли додано для зручності читання).

Шістнадцяткові константи позначаються префіксом 0x. До їх складу, окрім десяткових цифр можуть також входити шістнадцяткові цифри A , B, C, D, E, F для позначення чисел від 10 до 15. Наприклад, числу 0x12C відповідає десяткове число 300 ($1*16^2+2*16^1+12*16^0 = 256 + 32 + 12$).

Шістнадцяткові константи прийшли на зміну вісімковим. Їх також 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 (пробіли додано для зручності порівняння).

Двійкові константи позначаються префіксом 0b і можуть складатися тільки з одиниць та нулів. Наприклад, числу 0b1101 відповідає десяткове число 13 ($1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 8 + 4 + 0 + 1$).

1.3.4.2 Переведення десяткових цілих чисел в інші системи

Для переведення десяткового числа в іншу позиційну систему числення слід послідовно ділити число, а потім результат ділення, на основу нової системи і фіксувати залишки ділення, які представляються однією цифрою нової системи. Результатом буде число, яке записується цифрами залишків, починаючи з останнього

Приклад переведення числа 9001 у двійкову систему наведено в таблиці 1.2.

Так само можна переводити десяткове число до вісімкової системи шляхом ділення на 8, або до шістнадцяткової – діленням на 16.

Таблиця 1.2 – Переведення десяткового числа в інші системи числення

Число	9001	4500	2250	1125	562	281	140	70	35	17	8	4	2	1	0
Залишок від ділення на 2	1	0	0	1	0	1	0	0	1	1	0	0	0	1	<<
Результат	Двійкове представлення 0b10001100101001 По тріадах 0b 10 001 100 101 001 = 021451 По тетрадах 0b 10 0011 0010 1001 = 0x2329														

1.3.4.3 Константи дійсних чисел

Дійсні числа відрізняються від цілих тим, що мають дробову частину. У програмах на мові C/C++ такі числа можна записувати двома способами.

Перший спосіб – це запис з фіксованою крапкою. Це найбільш популярний спосіб, який знайомий усім, наприклад, 3.14, 245.127, 0.1. Але інколи такий спосіб не зовсім зручний, зокрема, якщо числа дуже маленькі, наприклад, 0.000000000000000002345.

Другий спосіб – це запис чисел з плаваючою крапкою. В цьому випадку число представляється у вигляді двох складових – мантиси і порядку. Наприклад, число 0.000000000000000002345 можна записати таким чином 2.345E-18. В цьому прикладі мантисою буде 2.345, а порядок дорівнює -18.

Мантиса представляє собою послідовність значущих цифр числа (початкові і кінцеві нулі, які входять до складу числа, не записуються). До складу мантиси входить також крапка, яка може бути розташована серед цифр де завгодно. Але найчастіше крапку ставлять після першої цифри. Таку мантису називають нормалізованою.

Порядок – це позитивне чи негативне ціле число, яке показує, на скільки порядків реальне число менше або більше мантиси. Тобто фактично запис 2.345E-18 означає $2.345*10^{-18}$.

Назва «з плаваючою крапкою» пов’язана з тим, що для отримання реального значення числа треба крапку перемістити на таку кількість розрядів, яка відповідає значенню порядку. Якщо порядок від’ємний крапка переміщується ліворуч, якщо додатній – праворуч.

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 12345000000000000 і 0.00000000012345 будуть представлені як 1.2345e +16 і 1.2345e-10. Це зручно як для запису чисел на папері, так і для зберігання чисел в пам’яті комп’ютера

У пам’яті комп’ютера дійсні числа зберігаються саме у формі з плаваючою крапкою. Але у програмах дотримуватися цього стандарту не обов’язково. Інколи зручніше записувати дійсні константи у звичайній формі, з фіксованою крапкою.

1.3.4.4 Символьні константи

Символьні константи – це літери, цифри, графічні символи, керуючі слеш-символи та ескейп-послідовності. У будь якому варіанті символьні константи розташовують між одинарними лапками. Наведемо приклади символьних констант: 'a' (літера а), '3' (символ цифри 3), '\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.

1.3.4.5 Рядки символів

Символьний рядок – це послідовність символів між подвійними лапками. Наприклад, "\t Лабораторна робота 1.\n". Якщо в програмі написано послідовно два рядки, вони розглядаються як один.

Докладніше про рядки символів можна почитати у підручнику [4] на сторінках 16-17.

1.3.5 Виведення та введення даних

Виведення і введення даних є не зовсім простим завданням. У підручнику [3] цій темі присвячено цілий розділ 5 і він стосується тільки мови С. Ви можете ознайомитися з цим розділом самостійно.

А поки що ми познайомимося з найбільш простими рішеннями цієї проблеми.

Заголовний файл бібліотеки С для потокового введення/виведення підключається за допомогою директиви #include <stdio.h>. Директива #include <iostream> використовується для підключення заголовного файлу аналогічної бібліотеки С++.

1.3.5.1 Виведення інформації на консоль

В мові С для виведення інформації на консоль є декілька способів. Мабуть, найбільш потужним інструментом є функція printf, але в загальному випадку потребує певних знань для успішного використання. Виключенням є виведення символьних рядків, наприклад, printf("\t Лабораторна робота 1.\n").

Для виведення символьних рядків можна використовувати і функцію puts, наприклад, puts("\t Лабораторна робота 1.\n").

В с++ для виведення інформації на консоль можна використовувати об’єкт cout разом з операцією виведення <<. В цьому разі, попередній приклад буде виглядати так: cout << "\t Лабораторна робота 1.\n".

Перевага такого способу полягає в тім, що він забезпечує просте виведення не тільки рядків символів, але й даних інших типів, наприклад, cout << "Сума "<< 3 << 'i' << 5 << " дорівнює " << (3+5) << endl.

В наведеному прикладі операція виведення << використовується послідовно декілька разів для того самого об’єкта cout. Крім того,

використовується константа endl, яка забезпечує виведення символу переходу на новий рядок.

В разі використання функції printf попередній приклад буде виглядати так: printf("Сума %di%d дорівнює %d\n", 3, 5, 3+5); Для початківця це виглядає трохи дивно.

З функцією printf ми познайомимося пізніше більш докладно, а поки що будемо використовувати puts або cout.

1.3.5.2 Підготовка для введення даних з консолі

Перед тим як знайомитися із засобами для введення даних треба з'ясувати, а куди ж ці дані будуть вводитися і де вони будуть зберігатися у програмі.

Для визначення місця зберігання даних у програмі використовується поняття «змінна». Така назва пов'язана з тим, що значення, яке знаходиться у «змінній» може мінятися. У змінної є ім'я, а для збереження даних, що пов'язані із змінною, в пам'яті комп'ютера виділяється певна ділянка пам'яті, розмір якої залежить від типу змінної. Для того, щоб пам'ять було виділено, змінну треба оголосити.

Оголошення змінної, яка буде зберігати цілі числа може виглядати так: int x1, де x1 – це ім'я змінної, а int – це один із можливих типів для цілих чисел. Схоже виглядає і оголошення змінної для дійсних чисел – float result, де result – це ім'я змінної, а float – це один із можливих типів для цілих чисел.

Трохи інакше оголошуються змінні, які пов'язані з рядками символів. В даному випадку програміст має визначити, скільки місця в пам'яті він замовляє для запису рядка. Оскільки рядок складається з символів, то для визначення типу використовується назва типу для символів – char, а після назви змінної в квадратних дужках записується кількість байтів потрібної пам'яті, наприклад, char str[20]. Тут оголошено рядок символів з ім'ям str, довжина якого не може перевищувати 19 символів (ще один символ – ознака кінця рядка).

1.3.5.3 Засоби для введення даних з консолі

В мові С найбільш потужним засобом введення інформації з консолі можна вважати функцію scanf, але її використання потребує певного рівня підготовки. Тому цю функцію поки що ми розглядати не будемо.

В с++ для введення інформації з консолі можна використовувати об'єкт cin разом з операцією введення >>. Так само, як і у випадку операції << з об'єктом cout, операція >> може використовуватися послідовно декілька разів для того самого об'єкта cin. Наприклад, якщо попередньо було оголошено змінні x1 та x2 (int x1, x2), то введення даних у ці змінні можна організувати так: cin >> x1 >> x2.

Особливість об'єкта cin полягає в тім, що він сприймає пробіл, як символ завершення вводу. З одного боку це добре, бо ми можемо одразу через пробіл

набрати два числа и натиснути Enter. З іншого боку це погано, бо робить неможливим введення рядків символів з пробілами.

Для введення рядків символів, які містять пробіли, можна використовувати функцію мови С gets. Наприклад, якщо було оголошено рядок символів str (char str[21]), то введення даних у цю змінну можна організувати так: gets(str).

Але недолік цієї функції полягає в тім, що вона не контролює кількість введених символів. Якщо користувач введе рядок довший, ніж ділянка виділеної пам'яті, то функція продовжить записувати інформацію за межами ділянки, і результат буде непередбачуваним.

Функція getline об'єкта cin також дозволяє вводити символьні рядки, що містять пробіли. До того ж ця функція дозволяє обмежити кількість символів, що буде прочитано із буфера. Тобто рядок з пробілами можна ввести таким чином: cin.getline(str,20).

1.3.5.4 Проблеми з введенням даних

Проблеми введення даних з консолі пов'язані з тим, що цей процес складається з двох етапів.

На першому етапі символи, які вводить користувач, потрапляють у буфер вводу (це ділянка пам'яті, яка не пов'язана з програмою).

Другий етап починається після того, як користувач натиснув клавішу Enter. Тут вже програма звертається до буфера і зчитує потрібні дані.

І тут може виникнути ситуація, коли в буфері залишається якась інформація, що може стати причиною некоректної роботи програми.

Розглянемо два найбільш поширених випадки.

Перший виникає, якщо після введення числа за допомогою cin іде введення рядка символів. Проблема полягає в тім, що cin залишає символ Enter у буфері. Внаслідок цього функція введення рядка, яка викликається після cin, зчитує цей символ і вважає, що введено пустий рядок.

Подолати цю проблему можна викликавши функцію fflush(stdin) або rewind(stdin) перед введенням рядка символів.

Другий проблемний випадок виникає тоді, коли користувач вводить більше даних, ніж потрібно програмі. Якщо далі в програмі ідуть оператори зчитування, то інформація буде продовжувати зчитуватися з буфера.

Найкращій спосіб вирішення цієї проблеми – виділяти достатньо пам'яті для рядка. Тут вже fflush(stdin) або rewind(stdin) не допоможе.

1.3.6 Використання математичних функцій

Для обробки даних в програмах можна використовувати бібліотеку математичних функцій, яка підключається за допомогою директиви #include <math.h> або #include <cmath>.

Для виклику функції необхідно записати її ім'я і далі в дужках, через кому, перерахувати необхідні параметри. Наприклад, щоб піднести змінну x1

до ступеня 1.3, достатньо записати pow (x1, 1.3).

У лістингу 1.2 наведено фрагмент коду з оголошенням змінних, введенням/виведенням даних та виконанням простих розрахунків

Лістинг 1.2 – Приклад програми обробки даних

```
puts("\n\tЗавдання 2");
 puts("Введення даних та обробка даних");
 //Оголошення змінних
 int x1, x2;
 double result;
 char fio[80]; // memory for 79 chars + '\0'
 //Введення даних
 puts("Введіть ім'я та прізвище студента");
 cin.getline(fio, 80);
 puts("Введіть два цілих числа через пробіл\n"
      "(рік та дату народження)");
 cin >> x1 >> x2;
 //Обробка та виведення результатів
 cout << "Студент " << fio << endl;
 result = x1 & x2; //побітове множення
 cout << x1 <<" & " <<x2 << " = " << result << endl;
 cout << "Кубічний корінь із " << result << " = "
      << cbrt(result) << endl;
```

1.4 Рекомендації до виконання роботи

1.4.1 Підготовка до роботи

1.4.1.1 Встановлення Code::Blocks

Щоб скачати .exe файл для встановлення Code::Blocks можна скористатися посиланням [4], або набрати у веббравзері запит download Code::Block і вибрати один із запропонованих сайтів.

Після завантаження .exe файлу рекомендується закрити всі застосунки, після цього запускаємо цей файл.

Натискаємо next, погоджуємося з умовами ліцензії та усіма пропозиціями стосовно конфігурації і натискаємо Instal.

Якщо виникають проблеми з установкою, то можна перейти за посиланням [5] і подивитись відео.

1.4.2 Створення найпростішого проєкту

Після запуску Code::Blocks має з’явитися головне вікно середовища, рисунок 1.3.

Щоб створити новий проект можна вибрати на головному екрані пропозицію «Create a new project». Можна також скористатися функціями головного менют File > New > Project....

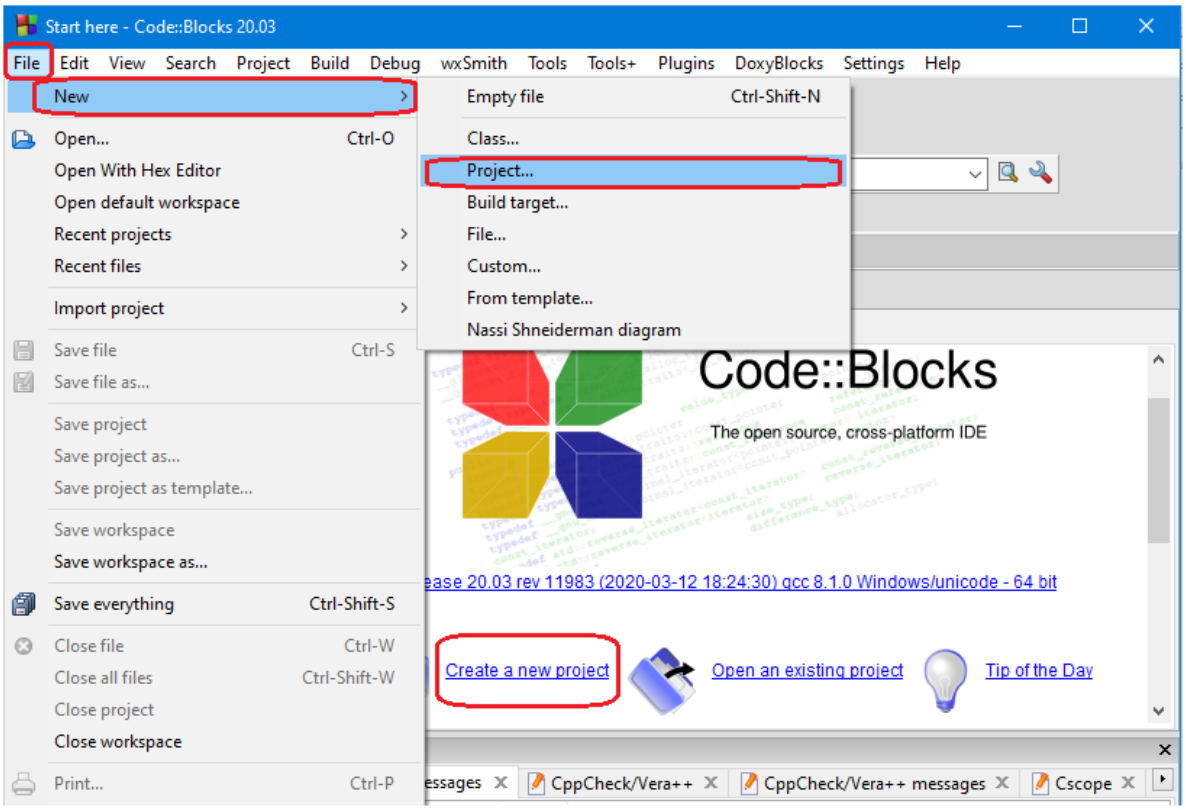


Рисунок 1.3 – Початкове вікно Code::Blocks

У новому вікні, що з’являється, рисунок 1.4, вибираєм Projects > Console Application >> Go.

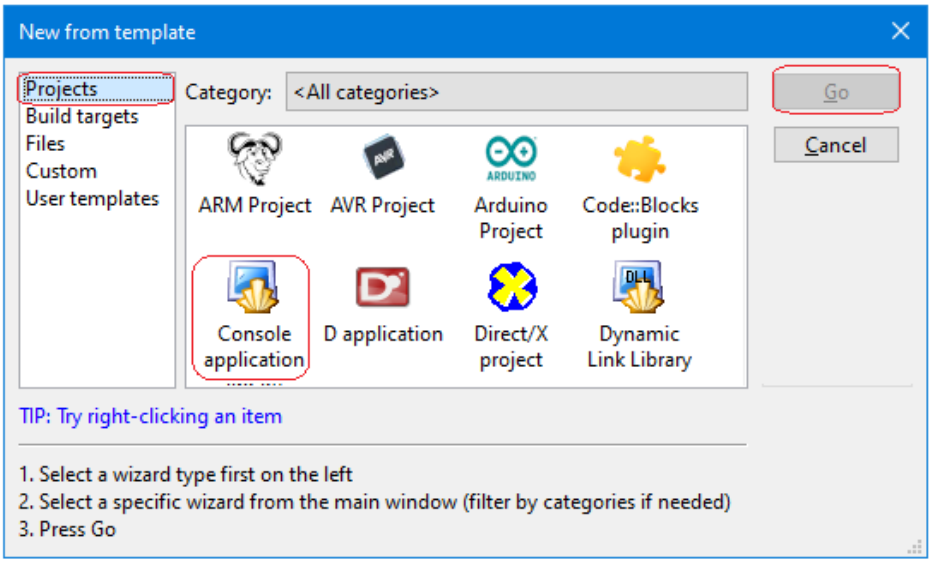


Рисунок 1.4 – Вибір типу проєкту

У наступному вікні, рисунок 1.5 просто натискаймо Next, щоб перейти до майстра створення консольного проєкту.

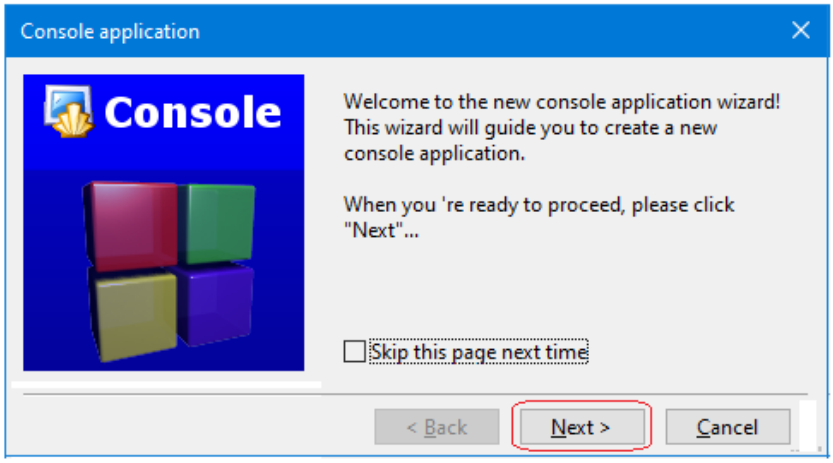


Рисунок 1.5 – Вікно привітання майстра консольних застосунків

В наступному діалозі треба вибрати мову програмування C++ і знов натиснути Next.

Далі треба ввести назву проєкту. То ж визначимося з назвами проєктів для лабораторних робіт. Будемо дотримуватися такого правила. Назва проєктів до лабораторних робіт мають складатися з прізвища студента, перших букв імені та по-батькові, суфікса lab та номера лабораторної роботи. Наприклад, проєкт може називатися ByvoinoPG_lab1.

Даному проєкту дамо таку саму назву, як для лабораторної роботи, тільки без номера роботи. Для наведеного прикладу це буде ByvoinoPG_lab.

Слід також визначитися з місцем зберігання проекту і знов натиснути Next.

В останньому діалозі погодитися із запропонованим компілятором і конфігурацією проєкту та натиснути кнопку Finish.

Після цього з’являється робоче вікно Code::Blocks, що схоже на те, яке було показано на рисунку 1.1. Для того, щоб побачити код програми треба відкрити папку Sources і двічі клацнути по файлу main.cpp.

Для запуску проєкту можна скористатися функцією головного меню середовища Build > Build and run.

Як результат має з’явитися консоль і на ній повідомлення з вітанням.

1.4.3 Деякі налаштування середовища розробки

Познайомимося з деякими можливостями налаштування середовища.

1.4.3.1 Редагування і налаштування української мови

На рисунку 1.1 код програми ми бачимо у вікні редактора тексту. Редагування не потребує спеціальних навичок.

Перекладіть, наприклад, текст привітання на українську мову.

Перед привітанням додайте виведення інформації про автора проєкту (прізвище, ім’я, групу).

Для того, щоб українські слова нормально відображалися на консолі, додайте до коду директиву #include <windows.h> , а на початку тіла функції main() додайте виклики системних функцій:

```
// Налаштування для української мови
SetConsoleCP(1251);
SetConsoleOutputCP(1251);
```

Після цього протестуйте програму.

1.4.3.2 Форматування тексту програми

Текст програми має бути обов'язково відформатований. Хоча компілятор не перевіряє форматування, форматування є дуже важливим елементом програмування. Докладну інформацію про форматування коду можна знайти за посиланням [6].

Але сучасні редактори в інтегрованих середовищах розробки програм дозволяють форматувати код автоматично. В Code::Blocks достатньо викликати контекстне меню редактора коду (права кнопка миші в межах поля редагування) і викликати функцію Format use AStyle (артистичний стиль).

Для тестування цієї можливості зсувайте рядки коду довільно праворуч і ліворуч, а потім викликайте цю функцію.

1.4.3.3 Пошук заміна в тексті

Для пошуку і замін в тексті програми можна скористатися функціями головного меню Search>Find та Search>Replace. Є також відповідні іконки на панелі інструментів і комбінації клавіш.

Для того, щоб опанувати ці операції реалізуймо виведення текстів за допомогою функції puts. Для цього комбінацію символів «cout << » змінимо на символи « puts(». А комбінацію « << endl» на символ закриваючої дужки.

Після редагування тексту перевірте працездатність програми.

1.4.3.4 Деякі налаштування редактора та меню

Розглянемо деякі налаштування редактора, що дозволять зручно формувати звіти з лабораторних робіт.

Перш за все налаштуємо шрифт. Для цього вибираємо функцію головного меню Setting >Editor. В діалозі, що з'явився, вибираємо сторінку GeneralSetting і відкриваємо закладку Editor Setting. Далі в групі налаштувань Font натискаємо кнопку Choose і налаштовуємо шрифт Courier New, звичайний, розмір 12.

Якщо розмір шрифту здається незручним, його представлення можна масштабувати. Для цього слід натиснути клавішу Ctrl і коліщатком миші налаштувати бажане представлення.

Далі налаштуємо праве поле у редактора, щоб контролювати ширину рядків коду. Вибираємо в налаштуваннях редактора сторінку Margins and caret і в групі налаштувань Right margin налаштовуємо Hint column = 60.

Після цього налаштування в полі редактора з'являється вертикальна лінія в заданій позиції. Точно це значення встановите після того, як будете переносити код до звіту.

Для того, щоб редактор не позначав українські слова як помилкові, можна вибрати функцію головного меню Plugins>Manage plugins… Далі в переліку плагінів знайти Spellchecker і вимкнути його за допомогою кнопки Disable.

Можна також вимкнути DoxyBlocks, яким ми користуватися не будемо. Це трошки скоротить головне меню.

1.4.3.5 Налаштування панелі інструментів

Під головним меню розташовано багато панелей інструментів, але деякі з них нам не знадобляться. Тому можна тимчасово відмовитися від їх відображення.

Для цього треба викликати функцію головного меню View>Toolbars і залишити тільки чотири опції – Code completion, Compiler, Main, Debugger.

Можна також викликати функцію Optimize toolbars.

Для того, щоб ці налаштування збереглись слід викликати функцію меню View>Perspectives>Save current.

1.4.3.6 Налаштування консолі

Запустіть програму на виконання. Як результат, на екрані з'явиться вікно консолі. Клацніть по іконці, що розташована у верхньому лівому куті консолі. У відповідь має з'явитися меню налаштувань консолі.

За допомогою функції «Розмір» налаштуйте ширину консолі відповідно до найбільшої ширини рядків, що будуть виводити наші програми. Це десь 65-70 символів шрифту Courier New.

За допомогою функції «Перемістити» розташуйте консоль у лівому верхньому куту екрану.

За допомогою функції «Властивості» викликайте діалог налаштувань шрифтів та кольорів і налаштуйте ці параметри на свій розсуд.

1.4.4 Шаблон для лабораторних проєктів

Усі проєкти, що будуть створюватися в межах лабораторних робіт, містять певну стандартну частину. Зокрема, це деякі директиви препроцесора, оголошення простору імен, функція main().

Крім того, функція main() має забезпечити введення/виведення інформації українською мовою. А також виводити на консоль інформацію про

розробника проєкту, номер лабораторної роботи і завдання, що вирішуються в проекті.

Тому доцільно створити проєкт, який буде містити цю стандартну частину і зберігати його як зразок. Далі цей проєкт можна використовувати як шаблон для конкретної лабораторної роботи.

1.4.4.1 Створення шаблону

Поміняйте текст файлу main.cpp за зразком рисунку 1.6.

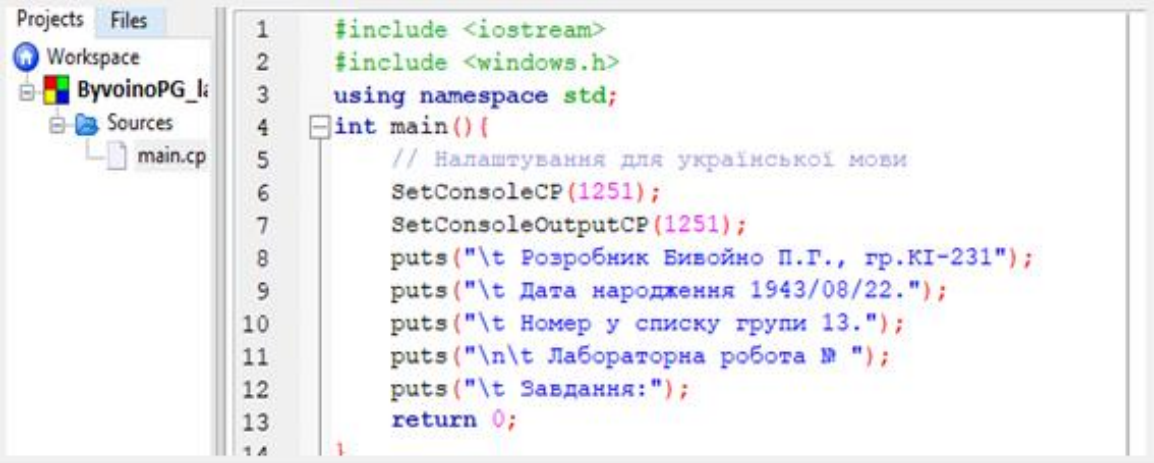


Рисунок 1.6 – Зразок файлу main.cpp для шаблону проєктів

Зверніть увагу на наявність правого поля, що позначає позицію, за якою не бажано друкувати текст.

1.4.4.2 Збереження шаблону

Для створення шаблону слід вибрати функцію головного меню File>Save Project as template. У діалозі, що має з’явитися, треба ввести назву шаблону і шаблон буде створено.

Назва шаблону може бути яка завгодно, а коли шаблон буде використовуватися для створення нового проєкту, то за замочуванням для нового проєкту буде пропонуватися ім’я, що співпадає з ім’ям проєкту, на основі якого створювався шаблон

Після створення шаблону проєкт можна закрити викликавши функцію Close project з контекстного меню проєкту.

1.4.4.3 Використання шаблону для створення проєкту

Щоб скористатися шаблоном треба викликати функцію створення нового проєкту і перейти до діалогу вибору шаблону для проєкту, рисунок 1.7. На панелі, що розташована ліворуч, вибираємо варіант User template. Після цього вибираємо потрібний шаблон із списку, що з’являється і натискаємо кнопку Go.

Далі слід визначитися з папкою, де буде зберігатися проєкт. Для цього краще створити нову папку з назвою, наприклад, Lab1.

Після вибору папки буде запропоновано визначитися з ім’ям проєкту.

Якщо файл шаблону створювався за рекомендаціями, то до імені достатньо буде додати номер лабораторної роботи.

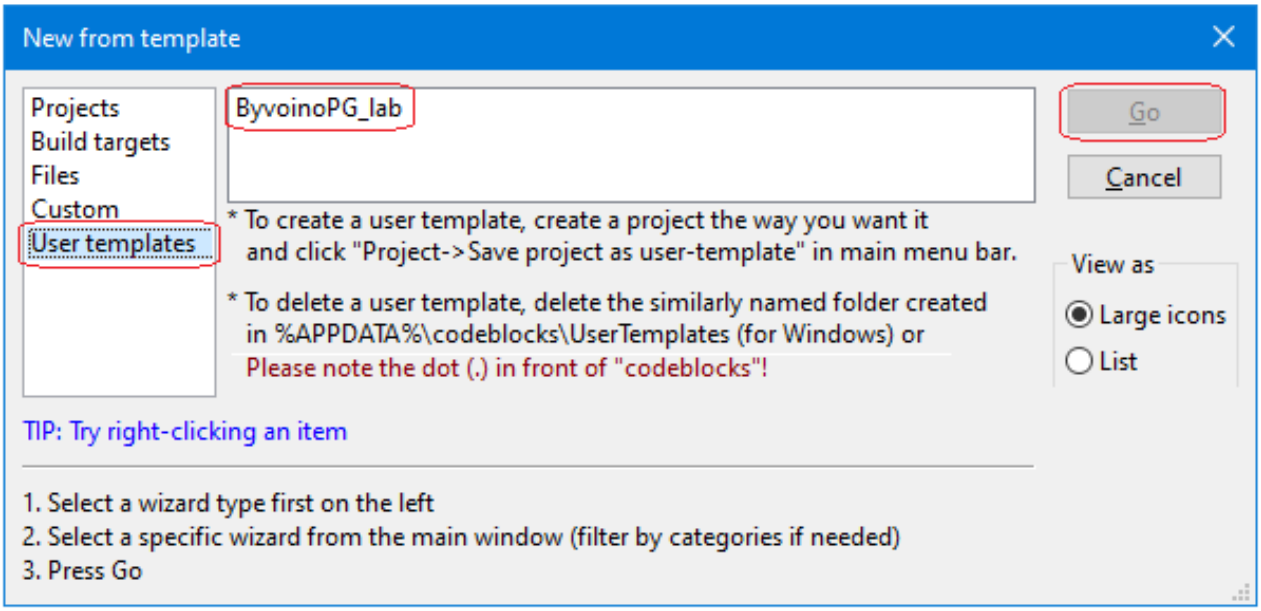


Рисунок 1.7 – Діалог вибору шаблона для проєкту

1.4.5 Реалізація тестування форм представлення чисел

Першим завданням на програмування в проєкті лабораторної роботи 1 буде створення тесту для перевірки правильності переведення цілого числа у двійковий, вісімковий та шістнадцятковий коди.

Для тестування використовувати число, яке є сумою року народження студента і дня народження

У лістингу 1.3 наведено приклад коду для перевірки результатів переведення. Об’єкт cout приймає числа і у двійковій, і у вісімковій, і у шістнадцятковій формах, але виводить їх за замовчуванням у десятковій.

Перед написанням програми слід виконати переведення цього числа спочатку у двійковий код, а потім цей код перевести у вісімковий (по тріадах) і шістнадцятковий (по тетрадах) .

Лістинг 1.3 – Код тестування правильності переведення числа 9001

```
puts("\t Лабораторна робота № 1");
puts("\t Завдання 1:");
puts("Створити програму тестування форм представлення"
     " цілих чисел");
puts("Результат тестування форм представлення числа 9001");
cout << "0b10001100101001 = " << 0b10001100101001 << endl;
cout << "021451 = " << 021451 << endl;
cout << "0x2329 = " << 0x2329 << endl;
```

Приклад переведення числа наведено в таблиці 1.2. Аналогічна таблиця має бути наведена у звіті з лабораторної роботи.

Написану програму слід запустити і пересвідчитися у правильності результату. Якщо результат не відповідає заданому числу, знайти помилку у переведенні і виправити її.

1.4.6 Введення даних та простими розрахунками

Наступне завдання з програмування – це введення тексту і двох чисел, обчислення і виведення результату заданої операції над числами та обчислення заданої функції для результату.

Функцію і операцію вибираємо з таблиці 1.1

Під час тестування програми в якості рядка символів треба вводити своє ім’я та прізвище, а для розрахунків вводити числа, що дорівнюють року і дню дати народження.

Приклад виконання схожого завдання наведено вище, в лістингу 1.2.

1.4.7 Експерименти з варіантами введення рядків символів

Після налагодження програми слід провести експерименти з дослідження можливостей введення рядків символів різними способами. Для цього зменшити обсяг пам’яті для змінної fio, наприклад, до 10 і спробувати вводити рядки меншої і більшої довжини. Пробіл всередині обов’язково має бути.

Далі замінити в коді cin.getline на функцію gets і провести такі самі експерименти, а потім на cin, і знов провести такі самі експерименти.

Про результати експериментів докладно написати у звіті.

1.4.8 Реалізація завдання з підручника

Виконати пункт 9 завдання до самоконтролю розділу 2 підручника[3], стр.30 (спробувати зрозуміти, що робить наведена програма).

Додати до проєкту наступне:

– включити перед функцією main() прототип функції FrameLine з пункту 9 підручника;

– включити до функції main(), як завдання 3, фрагмент коду з функції main() підручника, де використовується функція FrameLine;

– додати до файлу після функції main() код функції FrameLine;

– змінити ім’я функції FrameLine на frameLine скориставшись функцією Rename.

1.4.9 Експерименти з функцією з підручника

Виконати пункт 10 підручника і про результати експериментів докладно написати у звіті

1.5 Завдання для самостійної роботи

Опрацювати розділ 1 підручника [3] і підготуватися до відповідей на запитання для самоконтролю (стр.16) під час захисту роботи.

Опрацювати розділ 2 підручника [3]. Підготуватися до відповідей на запитання для самоконтролю (стр.30) під час захисту роботи

1.6 Вимоги до звіту

1.6.1 Основні вимоги до оформлення

Звіт оформлюється у вигляді текстового документа на сторінках формату А4 відповідно до вимог ДСТУ3008:2015, розділ 7 [2].

Шрифт (окрім лістингів коду) Times New Roman, 14, без курсиву і підкреслень. Для лістингів шрифт Courier New 12.

Поля: верхнє і нижнє від 1.5 до 2, ліве – 2.5, праве 1.5.

Нумерація сторінок праворуч, зверху.

Верхній колонтитул має містити Прізвище та ім’я студента, групу, номер у списку групи та дату народження у форматі рррр/мм/дд. Останні дані потрібні для формування і перевірки варіантів індивідуальних завдань.

Абзаци основного тексту вирівнюються по ширині. Перший рядок абзацу повинен мати відступ 1.25 або 1.3, між абзацами основного тексту інтервал не потрібен.

Текст має бути структурованим на розділи, підрозділи, пункти та, можливо, підпункти.

Для нумерації структурних елементів використовувати багаторівневий список. Після номера крапка не ставиться, роздільником має бути пробіл.

На рисунки, таблиці та лістинги в тексті мають бути посилання.

Рисунки, таблиці, лістинги нумеруються в межах розділу, тобто номер складається з двох чисел – номера розділу і порядкового номера рисунка з роздільником у вигляді крапки. Перед номером має бути слово «Рисунок » або «Таблиця », або «Лістинг » з пробілом. Після номера має стояти роздільник тире « – » і далі назва з великої літери.

Підпис рисунка розташовують під рисунком по центру. Назви таблиць і лістингів розташовують зверху, вирівнюють по лівому краю з абзацним відступом.

Рисунок має бути елементом тексту, вирівнюватися по центру, без абзацного відступу в мати інтервали відступу від попереднього тексту. Те саме стосується підпису під рисунком і назв таблиць і лістингів. Вони не повинні зливатися з основним текстом.

Заголовок розділу (назва лабораторної роботи) друкується великими жирними літерами і вирівнюється по центру без абзацного відступу.

Заголовки підрозділів, пунктів і підпунктів вирівнюються по лівому краю з абзацним відступом.

Усі заголовки повинні мати інтервал від попереднього тексту і перед наступним текстом.

1.6.2 Вміст звіту

Назва роботи (як заголовок розділу). Можна скопіювати з методички.

Мета роботи (як підрозділ 1.1). Текст цього підрозділу можна теж

Завдання на роботу (як підрозділ 1.2). Теж можна скопіювати з методички.

Результати виконання роботи, як заголовок підрозділу 1.3.

Програмна реалізація завдання на C/C++, як заголовок пункту 1.3.1. Окрім заголовка в цьому пункті має бути коротенький опис програми і посилання на лістинг 1.1 з кодом програми. Далі має йти сам лістинг.

Результати тестування програми, як заголовок пункту 1.3.2. Окрім заголовка в цьому пункті має бути коротенький опис процесу тестування та висновки про результатам тестування і посилання на лістинг 1.2 з результатами тестування (копія тексту консолі). Далі має йти сам лістинг.

Переведення цілого числа з десяткової до інших форм представлення, як заголовок пункту 1.3.3. В цьому пункті написати як формувалося число, відповідно до варіанта, і навести таблицю з результатами переведення.

Результати дослідження способів введення текстових даних, як заголовок пункту 1.3.4. В цьому пункті докладно описати результати дослідження способів введення текстових даних.

Результати експериментів з прототипом функції, як заголовок пункту 1.3.5. В цьому пункті описати самі експерименти та їх результати.

Висновки, як заголовок підрозділу 1.4. В цьому підрозділі треба коротенько сформулювати що було зроблено і які нові знання та навички Ви отримали.

В технічній документації висновки НЕ пишуть від першої особи, тобто слово «Я» не використовують.

Висновки треба нумерувати. Вирівнювання і форматування так само, як основний текст.

1.7 Контрольні питання

– Структура програми на С.
– Структура функції main().
– Найпростіший варіант написання функції main().
– Призначення директив #include <cmath> та #include <iostream>
– Етапи виконання програми на С.
– Як користуватися об’єктами cout та cin.
– Функції puts та gets.
– Що таке директива препроцесора?
– Робота з вікном редактора коду (продемонструвати можливості).
– Як зберегти проект.
– Як відкрити збережений проєкт.

2 ЛАБОРАТОРНА РОБОТА 2.
ТИПИ ДАНИХ ТА РОЗРАХУНКИ ЗА ФОРМУЛАМИ

2.1 Мета роботи

– Познайомитися з поняттям тип даних та типами даних мови С, С++.

– Познайомитися з арифметичними операціями над змінними різних типів.

– Навчитися записувати арифметичні вирази.

– Познайомитися з форматним виведенням даних.

– Створити додаток, в якому будуть реалізовані завдання, пов’язані з темою роботи ,

2.2 Завдання на лабораторну роботу

– Створити проєкт відповідно до шаблону.

– Протестувати результати виконання операцій над різними типами даних, див. підрозділ «Рекомендації до виконання роботи».

– Реалізувати розрахунки за формулами пункту 7 завдань для самоконтролю розділу 4 підручника [3], сторінка 65. У виразі 7.1 вважати, що квадрат косинуса тільки від x. У виразі 7.1 тангенс брати від усього виразу (u*x*x), а логарифм тільки від 2x і приводити до основи u.

– Реалізувати форматне виведення граничних констант для різних типів даних,

2.3 Теоретична частина

2.3.1 Змінні та 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.

2.3.2 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 [3], стр.31-37.

2.3.3 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.

2.3.3.1 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 ‘Y’* true є допустимим у С і його можна обчислити. Результатом буде число 89, бо true перетворюється у 1, а код символу ’Y’ дорівнює 89. 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.

2.3.3.2 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 слід використовувати двічі. У цьому разі розмір типу int збільшується у два рази. Якщо застосувати тільки раз, то це буде той самий int. Так історично склалось, бо колись розмір int був два байти, а зараз він має 4 байти.

Модифікатори long та int можна комбінувати з модифікатором unsigned.

2.3.3.3 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.

Константи граничних значень для дійсних типів визначені у файлі float.h.

З повним переліком цих констант та їх описом можна ознайомитися за посиланням [7, 8]. В таблиці 2.2 наведено лише ті з них, які знадобляться для виконання лабораторної роботи.

Таблиця 2.2 – Назви граничних константи для типів C++

Назва типу	Мінімальне значення	Максимальне значення
char	CHAR_MIN	CHAR_MAX
short	SHRT_MIN	SHRT_MAX
int	INT_MIN	INT_MAX
long long int	LLONG_MIN	LLONG_MAX
float	FLT_MIN	FLT_MAX
double	DBL_MIN	DBL_MAX
long double	LDBL_MIN	LDBL_MAX

2.3.3.4 Типи цілочислових констант

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.

2.3.3.5 Типи констант дійсних типів

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).

2.3.4 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.

2.3.4.1 Звичайні змінні

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 кваліфікатору перетворює змінну у константу. Нижче наведено приклад оголошення константи 2π.

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

2.3.4.2 Макроконстанти

У мові C++ існує можливість задавати константи за допомогою 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)
```

2.3.5 Операція розміру sizeof

Операція sizeof повертає обсяг пам'яті, яку займає або потребує операнд цієї операції, тобто розмір операнду. Результат операції має тип long long unsigned int. В ролі операнду 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 ім'я змінної;

– sizeof (вираз).

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

2.3.6 Арифметичні операції

Перелік арифметичних операцій С++ 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 С++

- унарний	Зміна знаку операнду
*	Множення
/	Ділення
%	Остача від цілочислового ділення
+	Додавання
-	Віднімання
<< n	Зсув ліворуч на n розрядів (множення на 2 n разів)
>> n	Зсув праворуч на n розрядів (ділення на 2 n разів)

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.

2.3.7 Операції присвоєння

2.3.7.1 Проста операція присвоєння

Операція присвоєння є однією із основних операцій у будь якій мові програмування. У мові С++ 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.

2.3.7.2 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 знаку 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.

2.3.7.3 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.

2.3.8 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

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

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), 2.0) + b)
$e^{2.5x}$	exp(2.5*x)
$a^{1.5+b}$	pow(a, 1.5+b)

При обчисленні 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.

2.3.9 Пріоритети операцій у C++

У таблиці 2.5 наведено пріоритети 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

2.3.10 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.

2.3.10.1 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, а 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.

2.3.10.2 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.

2.3.10.3 Явне перетворення типів

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

(*тип*) *вираз*

2.3.11 Бібліотека математичних функцій cmath

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

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
cbrt(вираз)	Кубічний корінь
pow(вираз, ступінь)	Піднесення до ступеню
exp(вираз для степеню числа e)	Експонента
log(вираз)	Натуральний логарифм
sin(вираз)	Синус
cos(вираз)	Косинус
tan(вираз)	Тангенс
asin(вираз)	Арксинус
acos(вираз)	Арккосинус
atan(вираз)	Арктангенс
ceil(вираз)	Округлення до найближчого більшого
floor(вираз)	Округлення до найближчого меншого

Звернення до 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). Приклади використання функцій можна знайти в інтернет, зокрема за посиланням [9].

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.

2.3.12 Форматне виведення даних через функцію printf()

У тих випадках, коли результати обчислень потрібно виводити певним чином (вирівнювання рядків, потрібна точність і т.і.), використовують функцію printf(), прототип якої визначено у файлі stdio.h.

```
int printf(const char *format, arg-list)
```

Як бачимо, ця функція має два параметри.

Перший параметр – це рядок символів format, у якому спеціальним способом вказується, яким чином потрібно виводити інформацію.

Другий параметр – це список arg-list, в якому через кому перелічені дані, які потрібно виводити. Ці дані можуть бути константами, змінними, виразами.

Тобто перший параметр можна назвати «як», а другий – «що».

Рядок, на яку вказує format, складається з об’єктів двох різних призначень. По-перше, це символи, які самі повинні бути виведені на екран. По-друге, це специфікатори формату, що визначають вид, в якому будуть виведені аргументи зі списку arg-list.

У таблиці 2.7 наведено деякі коди форматування.

Специфікатор формату починається з символу %, за яким іде код формату. Кількість аргументів у списку даних має точно відповідати кількості специфікаторів формату, причому слідувати вони повинні в однаковому порядку.

Наприклад, після виконання такого рядка коду:

```
printf("Сума чисел %d та %d дорівнює %d\n", 10, 5, 10 + 5);
```

на консолі з’явиться текст «Сума чисел 10 та 5 дорівнює 15».

Функція printf () повертає кількість дійсно виведених символів. Від’ємне значення означає помилку.

Команди форматування можуть містити модифікатори, які означають ширину поля, точність і прапор вирівнювання вліво.

Змінна цілого типу, вміщена між символом % і командою форматування, працює як специфікатор мінімальної ширини поля, заповнюючи поле виведення пробілами або нулями так, щоб забезпечити зазначену мінімальну ширину. Якщо рядок або число більше, ніж цей мінімум, вони будуть повністю виведені. За замовчуванням заповнення робиться пробілами. При виведенні числових змінних, якщо потрібно заповнення

нулями, поміщається нуль перед специфікатором мінімальної ширини поля. Наприклад,%05d буде доповнювати числа, що складаються з менш ніж 5 цифр, нулями до п’яти цифр.

Таблиця 2.7 – Коди форматування функції printf()

Код	Аргумент
%c	Символ типу char
%d	Десяткове число цілого типу зі знаком
%lld	Десяткове число цілого типу long long зі знаком
%e	Наукова нотація (e нижнього регістру)
%f	Десяткове число з плаваючою точкою
%g	Використовує код% e або% f - той з них, який коротше
%Lg	Для типу long double (або %Le, або %Lf)
%o	Вісімкове ціле число без знаку
%s	Рядок символів
%u	Десяткове число цілого типу без знаку
%llu	Десяткове число цілого типу long long без знаку
%x	Шістнадцяткове ціле число (літера x нижнього регістру)
%llx	Те саме для типу long long
%p	Вказівник на якийсь елемент
%*	Вказівник на значення, яке буде підставлено у формат
%%	Вивести знак %

Результат використання модифікатора точності залежить від типу команди форматування. Наприклад, %10.4f означає виведення числа шириною мінімум 10 символів з чотирма знаками після крапки. Однак при використанні спільно з специфікаторами g або G модифікатор точності задає максимальну кількість значущих цифр.

Коли модифікатор точності застосовується до цілих чисел, він вказує мінімальну кількість цифр. (У разі необхідності відображаються попередні нулі.)

Коли модифікатор точності застосовується до рядків, число після десяткового дробу вказує максимальну довжину поля. Наприклад,%5.7s виводить рядок довжиною не менше п’яти і не більше семи символів. Якщо рядок довший, ніж максимальна ширина поля, то останні символи будуть урізані.

За замовчуванням дані виводяться з вирівнюванням вправо. Це означає, що якщо ширина поля більше, ніж дані, що виводяться, то дані будуть розміщені на правому краю поля. Можна задати режим вирівнювання вліво, вставивши знак мінус відразу після знаку відсоток. Наприклад,% -10.2f притисне вліво число з плаваючою точкою з двома знаками після коми в полі з десяти знаків.

Є два специфікатор формату, що дозволяють printf () відображати цілі числа типу short і long. Ці специфікатор можуть застосовуватися спільно з специфікаторами типу d, i, o, u, x і X. Специфікатор l «говорить» printf () про те, що далі йдуть дані типу long. Наприклад,% ld означає, що потрібно вивести long int.

Модифікатор l може бути також перед специфікаторами типу з плаваючою точкою e, E, f, g і G, вказуючи, що далі йде змінна long double.

Символ # має особливе значення, коли використовується з деякими специфікаторами формату функції printf(). Якщо поставити # перед специфікаторами g, G, f, e або E, то десяткова точка буде ставитися навіть за відсутності цифр після коми. Якщо поставити # перед специфікатором формату x, то шістнадцяткове число буде виведено з префіксом 0x. Використання # зі специфікатором o приводить до виведення префіксу 0. З іншими специфікаторами формату символ # використовуватися не може.

Специфікатори мінімальної ширини поля і точності можуть бути представлені не тільки у вигляді констант, а й за допомогою аргументів. Це робиться за допомогою символу *, який вставляється замість специфікатору У цьому випадку специфікатор приймає значення відповідного аргументу.

Докладніше із форматним виведенням та введенням можна познайомитися у розділі 5 підручника [3].

2.4 Рекомендації до виконання роботи

Створити проєкт для лабораторної роботи за допомогою шаблону. Додайте директиви включення заголовних файлів `<cmath>`, `<climits>`, `<float.h>`.

2.4.1 Програма реалізації операцій з даними різних типів

На цьому етапі потрібно реалізувати деякі арифметичні операції з даними різних типів, а саме:

– Операцію додавання символу з числом, (символом має бути перша буква прізвища, а числом – день народження.

– Операцію додавання до року народження результату операції порівняння < для дня народження і місяця народження.

– Операцію цілочисельного ділення року народження на суму дня і місяця народження.

– Операцію отримання залишку від цілочисельного ділення року народження на суму дня і місяця народження.

– Операцію не цілочисельного ділення року народження на суму дня і місяця народження.

– Операцію зсуву року народження ліворуч на 2 розряди.

– Операцію зсуву року народження праворуч на 2 розряди.

Реалізація цього завдання може виглядати як лістинг 2.1.

Лістинг 2.1 – Реалізація операції з даними різних типів

```
puts("\t Завдання 1");
puts("Реалізувати деякі арифметичні операції\n"
     " з даними різних типів");
printf(" 'B' + 22 = %d\n",'B' + 22 );
printf(" 'B' + 22 = %c\n",'B' + 22 );
printf(" 1943 + (22 > 8) = %d\n", 1943 + (22 > 8));
printf(" 1943 / (22 + 8) = %d\n", 1943 / (22 + 8));
printf(" 1943 %% (22 + 8) = %d\n", 1943 % (22 + 8));
printf(" 1943.0 / (22 + 8) = %f\n", 1943.0 / (22 + 8));
printf(" 1943 << 2  = %d\n",1943 << 2);
printf(" 1943 >> 2  = %d\n",1943 >> 2);
```

Потрібно модифікувати цей код відповідно до свого варіанта, отримати результат, який також навести у звіті.

Крім того, у звіті **дати пояснення**, чому результат саме такий.

2.4.2 Реалізація розрахунків за формулами

Завдання полягає в реалізації обчислення за формулами з розділу 4 підручника, сторінка 65, пункт 7. Початок розрахунків може виглядати так, як показано в лістингу 2.2.

В лістингу 2.3 наведено результати розрахунків за цими формулами, які були отримані для значень a = 1, u = 2, z = 3, y = 4, x = 5.

Лістинг 2.2 – Початок ділянки коду з розрахунками за формулами

```
puts("\n\t Завдання 2");
puts("Розрахунки за формулами");
//Оголошення змінних
double a, u, z, y, x;
//Введення даних
puts("Введіть a - номер у списку, u - варіант,\n"
     " z - день народження, y - місяць народження,\n"
     " x - місяць народження / 10");
cin >> a >> u >> z >> y >> x;
```

Після написання коду протестуйте програму для вхідних даних 1, 2, 3, 4, 5 і порівняйте з лістингом 2.3. Якщо результати не співпадають, то перевірте правильність програмування виразів. Різниця можлива тільки за рахунок різної кількості цифр і округлення.

Після перевірки введіть дані відповідно до свого варіанту, як написано в лістингу 2.2, і зафіксуйте результати для звіту.

Лістинг 2.3 – Результати контрольних обчислень за формулами

```
a=1.0, u=2.0, z=3.0, y=4.0, x=5.0
Вираз 7.1 = -1.89103
Вираз 7.2 = 0.718714
Вираз 7.3 = 19.0526
Вираз 7.4 = -0.71752
```

2.4.3 Форматне виведення розміру основних типів даних та їх граничних значень

Завдання полягає у виведенні на консоль інформації про основні типів даних у вигляді відформатованої таблички, лістинг 2.1.

Лістинг 2.1 – Інформація про основні типи даних

```
Тип          Розмір  Мін                    Макс
char              1 -128                   +127
short             2 -32768                 +32767
int               4 -2147483648            +2147483647
long long int     8 -9223372036854775808 +9223372036854775807
float             4  1.17549e-38            3.40282e+38
double            8  2.22507e-308           1.79769e+308
long double      16  3.3621e-4932           1.18973e+4932
```

Для реалізації завдання використовувати функцію printf. Для налаштування специфікаторів використовуйте інформацію з пункту 2.3.12 і будьте уважні до типів констант, вибирайте специфікатори з таблиці 2.7. Назви констант наведені в таблиці 2.2. Візьміть до уваги, що операція sizeof повертає результат типу long long unsigned int.

2.5 Завдання для самостійної роботи

Опрацювати розділи 3 та 4 з підручника.

Знайти відповіді на питання 1-12 розділу 3 підручника [3], стр.42 і підготуватися до відповіді на ці питання під час захисту роботи..

2.6 Вміст звіту

Назва роботи (як заголовок розділу). Можна скопіювати з методички.

Мета роботи (як підрозділ 2.1). Текст цього підрозділу можна теж можна скопіювати з методички.

Завдання на роботу (як підрозділ 2.2). Теж можна частково скопіювати з методички без зайвої інформації.

Результати виконання роботи, як заголовок підрозділу 2.3.

Програмна реалізація завдання, як заголовок пункту 2.3.1. Окрім заголовка в цьому пункті має бути коротенький опис програми і посилання на лістинг 2.1 з кодом програми. Далі має йти сам лістинг.

Результати тестування програми, як заголовок пункту 2.3.2. Окрім заголовка в цьому пункті має бути коротенький опис процесу тестування та висновки про результатам тестування із лістингом 2.2 з результатом контрольних розрахунків за формулами. Далі посилання на лістинг 2.3 з результатами тестування (копія тексту консолі) і сам лістинг.

Результати дослідження операцій з даними різних типів, як заголовок пункту 2.3.3. В цьому пункті надати докладні пояснення стосовно кожного рядка результатів з операціями над даними.

Висновки, як заголовок підрозділу 2.4. В цьому підрозділі треба коротенько сформулювати що було зроблено та які нові знання та навички отримано.

2.7 Контрольні питання

- Змінні і константи.
- Тип даних.
- Арифметичні типи даних.
- Описи змінних.
- Операція розміру sizeof.
- Арифметичні операції.
- Операції присвоєння.
- Побітові операції
- Вирази.
- Пріоритети операцій у C++.
- Узгодження типів.
- Бібліотека математичних функцій cmath.
- Використання printf() для виведення даних.

3 ЛАБОРАТОРНА РОБОТА 3. ФУНКЦІЇ

3.1 Мета роботи

– Познайомитися з поняттям функції у мові С, С++.
– Познайомитися з поняттям прототип функції.
– Створити програму з використанням функцій.
– Дослідити способи передачі параметрів у функції.

3.2 Завдання на лабораторну роботу

– Створити за шаблоном новий проєкт.
– Вибрати формулу для розрахунків з таблиці 3.1 і відповідно до варіанту виконати контрольний розрахунок.

Таблиця 3.1 – Завдання для проекту

Варіант	Формула
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	$f = a \cdot \sin^4(w \cdot x) - q\sqrt{w/x}$
4	$g = a \cdot e^{-b \cdot x} - b \cdot x \cdot e^{-a} \cdot \frac{\sin(c \cdot x)}{c}$
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	$\mu = \frac{ctg(d \cdot w)}{fw + d}\sqrt{\left\lvert \frac{f + d \cdot w}{ctg(f \cdot w)} \right\rvert}$
10	$y = a \cdot x^3 - b \cdot \sin(x) + \sqrt{\frac{b \cdot \sin(x)}{a}}$

– Написати функцію, яка повертає результат розрахунків через return.
– Написати функцію для розрахунків за тією самою формулою, яка повертає результат через параметр.

– Доповнити функцію main() операторами введення даних, звернення до написаних функцій та виведенням результатів. Функцію main() розташувати перед написаними функціями.

– Дослідити спосіб передачі параметрів by value.

– Дослідити спосіб передачі параметрів by reference.

3.3 Теоретична частина

Функції є основними будівельними блоками програми, написаної на мові С. Це оформлена стандартним чином ділянка 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.

3.3.1 Правила написання функцій

Всі функції мови С мають однакову структуру, що виглядає таким чином, рисунок 3.1.

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

Рисунок 3.1– Структура функції в мові С/С++

Тип значення, що повертається, це будь-який тип, що можливий у мові С. 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 показано на рисунку 3.2:

```
return  вираз ;
```

Рисунок 3.2 – Синтаксис оператора return

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, у лістингу 3.1 наведено опис функції, що повертає суму двох чисел.

Лістинг 3.1 – Приклад функції, що повертає суму двох чисел

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

У тілі функції визначено локальну змінну y, 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, лістинг 3.2.

Лістинг 3.2 – Інший варіант функції, що повертає суму двох чисел

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

3.3.2 Виклик функції

Код функції виконується тільки тоді, коли здійснюється звертання до неї. Таке звертання називають викликом функції. Виклик функції виглядає так, як показано на рисунку 3.3.

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

Рисунок 3.3 – Синтаксис звернення до функції

Список фактичних параметрів являє собою послідовність імен змінних або 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 операнду виразу. Значенням такого операнду буде

значення, що повертає функція.

Приклад звернень до функції, опис якої наведено у лістингах 3.1 та 3.2 пункті, представлено лістингу 3.3.

Лістинг 3.3 – Приклади звернення до функції

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

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 буде 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/3, а результатом буде число 5.5.

3.3.3 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_що_повертається ім'я_функції(список_формальних_параметрів);
```

Рисунок 3.4 – Структура прототипу функції

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);
```

3.3.4 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 с. 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 С++, за звичай вимагає включення у програму директиви використання простору імен:

```
using namespece std;
```

Тут std – це назва простору імен.

3.3.5 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 &.

3.3.5.1 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 лістингах 3.1, 3.2 з пункту 3.3.1, параметри до функції передаються по значенню.

3.3.5.2 Передача параметрів через посилання

У разі передачі параметрів через посилання до функції передаються посилання на фактичні параметрів (посилання – це фактично адреса параметра в пам’яті).

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 через return може повернути тільки одне значення.

Розглянемо приклад використання передачі параметрів через посилання для повернення результатів роботи функції, лістинг 3.4.

У наведеній нижче функції фактичні параметри обміняються значеннями.

Лістинг 3.4 – Передача параметрів за посиланням

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

Для тестування такої функції можна скористатися такою функцією main(), лістинг 3.5.

Лістинг 3.5 – Функція для тестування функції обміну даними

```
int main() {
    int x =10, y = 20;
    swap(x, y);
    cout << x << y << endl;
}
```

3.3.6 Області оголошення та доступу до імен

Область оголошення – це частина програми, в якій здійснюються 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.

3.3.6.1 Глобальні та локальні змінні

Найширшою областю оголошення є файл, який містить нашу програму. Змінні та константи, що оголошені у файлі 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. 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 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.

3.3.6.2 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.

3.3.6.3 Специфікатор static

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.

```
#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() << endl; goto m;
      return (0);
}
```

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.

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 **static**.

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

3.4 Рекомендації до до виконання роботи

Як приклад, ми розглянемо створення функцій для розрахунків за формулою 3.1.

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

3.4.1 Аналіз завдання та контрольний розрахунок

3.4.1.1 Аналіз

Аналіз формули показує, що в ній тричі використовується вираз *ax*, тому доцільно його обчислити окремо. Для спрощення перевірки отриманих результатів будемо обчислювати окремо чисельник і знаменник. Вже після цього виконаємо повний розрахунок

3.4.1.2 Контрольний розрахунок

Контрольний розрахунок виконується для того, щоб порівняти результати, які видає програма з результатом, який отримав програміст виконуючи розрахунки вручну.

Для контрольного розрахунку потрібно підібрати значення змінних, які дозволять обчислити результат без зусиль, можливо навіть без калькулятора.

Приймемо, що *ax* = 0.5, тоді *sin(ax)* теж буде приблизно 0.5, а чисельник буде приблизно 0.625.

Для обчислення знаменника приймемо, що b = 2, а x = 4. Тоді виходить, що a = 0.125, а вираз під коренем буде дорівнювати 0.1. Знаменник буде дорівнювати приблизно 0.31.

Таким чином, для значень a = 0.125, b =2, x = 4 результат має дорівнювати приблизно 2. Контрольні значення проміжних результатів наступні: *ax* = 0.5, чисельник ~ 0.625, знаменник ~0.31.

3.4.2 Створення проєкту

Створімо за допомогою шаблону новий проєкт. Додаймо директиву підключення бібліотеки математичних функцій.

3.4.2.1 Прототипи функцій

Перед функцією main() додаймо прототипи функцій, які будемо створювати. Для прикладу, що розглядається, вони будуть виглядати так, як показано в лістингу 3.6

Лістинг 3.6 – Прототипи функцій проєкту

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

До першої і другої функції значення a, b, x передаються за значенням.

Перша функція буде повертати результат через return.

Друга функція поверне результат через параметр y, який передається до функції за посиланням.

3.4.2.2 Часткова реалізація функції main()

Далі доповнимо функцію main() операторами введення даних, звернення до функції f1 та виведенням результату.

Частина реалізації функції main() для прикладу, що розглядається, наведено в лістингу 3.7.

Слід зазначити, що після написання цього варіанту головної функції запускати її на виконання нема сенсу, бо ще нема реалізації функції f1.

Лістинг 3.7 – Частина реалізація функції main() з викликом f1

```
puts("\n\t Завдання 1");
puts("    Створити і протестувати функції з отриманням\n"
       "результату через return і через посилання");
puts("Введіть дані (a, b, x) для розрахунку за формулою");
double a, b, x, y1;
cin >>a >>b >> x;

y1 = f1(a, b, x);
printf("Результат отриманий через return\n y1 = %f\n", y1);
```

3.4.2.3 Реалізація функції з поверненням результату через return

Відповідно до рекомендацій, які були сформульовані на етапі аналізу реалізуймо функцію f1. Код функції для прикладу, що розглядається, наведено в лістингу 3.8.

Лістинг 3.8 – Функція з поверненням результату через return

```
double f1(double a, double b, double x){
  double ax = a*x;
  double ch = ax + pow(sin(ax), 3.0);
  double zn = pow(ax/(1+x),1/b);
  double y = ch/zn;
  return y;
}
```

Тепер можна компілювати проєкт, виправляти помилки, якщо вони є, і запускати на виконання.

Спочатку слід ввести дані, які використовувалися для контрольного розрахунку. Якщо програма видає результат, що не відповідає очікуваному, слід знайти помилки. Вони можуть бути як в програмі, так і в розрахунках.

З методикою тестування роботи програми і переглядом результатів проміжних розрахунків ми познайомимося, в пункті 3.4.3.

3.4.2.4 Остаточна реалізація функції main() для даного завдання

Продовжимо реалізацію функції main(). Для цього додаймо ще оголошення змінної y2, посилання на яку будемо передавати до функції f2.

Додаймо також звернення до функції f2, та виведення результату роботи цієї функції.

Відповідний фрагмент реалізацію функції main() для прикладу, що розглядається, наведено в лістингу 3.9.

Зверніть увагу на різницю звернень до функцій f1 та f2. Друга функція не повертає результат через return, вона записує його за посиланням, яке передано через параметр функції.

Лістинг 3.9 – Фрагмент реалізація функції main()

```
puts("Введіть дані (a, b, x) для розрахунку за формулою");
double a, b, x, y1, y2;
cin >>a >>b >> x;
y1 = f1(a, b, x);
cout << "y1 = " << y1 << endl;
f2(a, b, x, y2);
cout << "y2 = " << y2 << endl;
return 0;
```

Але для того, щоб ця версія функції main() змогла працювати, потрібно реалізувати функцію f2.

3.4.2.5 Реалізація функції з поверненням результату через посилання

Реалізація функції f2 багато в чому повторює реалізацію функції f1, лістинг 3.10. Різниця полягає в тім, що в ній нема оператора return, натомість, результат присвоюється параметру, який містить посилання на змінну, яка знаходиться у функції main(), що викликає дану функцію.

Лістинг 3.10 – Функція, що повертає результат через параметр

```
void f2(double a, double b, double x, double &y){
  double ax = a*x;
  double ch = ax + pow(sin(ax), 3.0);
  double zn = pow(ax/(1+x),1/b);
  y = ch/zn;
}
```

Тепер можна запустити програму і в разі успішного виконання зберегти код і результат для звіту.

3.4.3 Робота з програмою в режимі налагодження

Режим налагодження дозоляє покрокове виконання програми або від однієї заданої точки до іншої. При цьому можна відстежувати значення усіх змінних програми.

Режим покрокового виконання можливий, якщо на панелі інструментів Compiler (там де зелений трикутник запуску) виставлено режим Debug (не Release).

3.4.3.1 Покрокове виконання програми.

Встановіть курсор на початку рядка з другим викликом функції puts. Після цього активізуйте функцію головного меню Debug>Run to cursor.

На екрані має з’явитися консоль з першим рядком тексту, який виводить програма і вікно Watches.

Для зменшення розмірів головного вікна доцільно відключити ліву панель головного вікна (View>Manager) і після цього розташувати головне вікно з панелями інструментів і кодом у правій половині екрану.

Вікно Watches розташувати ліворуч знизу, а консоль ліворуч зверху. Бажано, щоб вікна не перекривалися.

Вікно Watches показує, що змінні нашої програми поки що містять сміття.

Для руху далі по програмі можна використовувати функцію меню Debug>Next line або скористатися відповідною іконкою на панелі інструментів Debug.

Увага! Коли дійдете до введення даних, прослідкуйте, щоб курсор знаходився в межах консолі. Інакше можна пошкодити код програми.

Коли курсор перемітиться до рядка із зверненням до функції слід вибрати функцію Debug>Siep into. У цьому разі можна увійти в середину функції, що викликається, і слідкувати за її роботою. На цьому етапі можна побачити значення проміжних результатів розрахунків і порівняти їх з результатами контрольних розрахунків.

Зафіксуйте для звіту копія вікна Watches із значення проміжних результатів розрахунків.

Для руху всередині функції використовуємо Debug>Next.

Щоб завершити процес налагодження слід переключитися в режим Release на панелі інструментів Compiler.

3.4.3.2 Використання точок переривання

Зазвичай нема необхідності покрокового перегляду всієї програми. Тому частіше використовується переміщення по програмі від однієї контрольної точки до іншої. Такі контрольні точки називають ще точками переривання (breakpoins).

Щоб встановити точку переривання достатньо клацнути мишею ліворуч від рядка з кодом. В результаті на цьому полі має з’явитися червоний кружок.

Перейти до наступної точки переривання можна командою меню Debug>Start/Continue. Після зупинки на контрольні точці можна виконувати покрокове переміщення або переміщення до курсору.

3.4.4 Дослідження передачі параметрів за значенням

Додайте до функції f1 рядок коду, в якому присвойте нульові значення усім формальним параметрам що передаються до функції за значенням і

розташуйте цей рядок перед оператором return, після обчислення кінцевого результату.

Налаштуйте точку переривання перед цим рядком, а другу точку переривання налаштуйте в середині функції f2.

Запустіть програму в режимі налагодження і переконайтеся в тім, що те, що формальні параметри отримали нульові значення, не вплинуло на значення фактичних параметрів у функції main(). Доказом є те, що функція f2 отримала не змінені значення параметрів.

3.4.5 Дослідження передачі параметрів за посиланням

Приберіть значок & перед змінною, що повертає результат обчислень, в оголошенні і реалізації функції.

Налаштуйте точку переривання перед рядком з обчисленням результату. Переконайтесь в тім, що результат правильний.

Поясніть, чому функція не повернула результат.

3.5 Завдання для самостійної роботи

Треба написати s протестувати функцію, яка змінює значення трьох змінних цілого типу x1, x2, x3 таким чином, щоб виконувалась умова x1<= x2<=x3.

В якості тестового набору даних для цього завдання будемо вводити рік народження, місяць народження і день народження. Якщо місяць і день співпадають, день збільшити на 1.

3.5.1 Рекомендації до виконання завдання

Реалізацію завдання можна почати з доопрацювання функції main(), де викликати функцію тестування для шести можливих варіантів розташування вихідних даних, лістинг 3.11.

Лістинг 3.11 – Частина функції main() для завдання 2

```
puts("\n\t Завдання 2");
puts("    Написати функцію arrange з параметрами x1, x2, x3,\n"
     "яка змінить їх значення відповідно умові x1<=x2<=x3");
puts("Введіть  x1 = рік народження, x2 = місяць народження,\n"
     "x3 = день народження");
int x1, x2, x3;
cin >>x1 >> x2 >> x3;
testArrange(x1, x2, x3);
testArrange(  ,  ,   );
testArrange(  ,  ,   );
testArrange(  ,  ,   );
testArrange(  ,  ,   );
testArrange(  ,  ,   );
```

У наведеному фрагменті функції main() шість разів викликається функція testArrange, до якої за значенням передається три параметра в усіх можливих комбінаціях розташування..

Завдання цієї функції – надрукувати значення цих параметрів, потім викликати функцію впорядкування, після чого знов надрукувати значення параметрів. Код цієї функції наведено в лістингу 3.12.

Лістинг 3.12 – Функція тестування функції arrange

```
void testArrange(int x1, int x2, int x3) {
    printf("До x1=%d, x2=%d, x3=%d; ", x1, x2, x3);
    arrange(x1, x2, x3);
    printf("Після x1=%d, x2=%d, x3=%d;\n", x1, x2, x3);
}
```

Наступним кроком має бути реалізація функції arrange, але писати її в напряму, без допоміжних інструментів, не варто, адже існує шість можливих комбінацій розташування чисел і в аналізі цих комбінацій легко заплутатися.

Тому спочатку створімо функцію впорядкування двох чисел, лістинг 3.13. До функції передаємо посилання на дві змінні a та b. Якщо b<a то треба щоб змінні обмінялися значеннями. Для перевірки умови і прийняття рішення про обмін у функції використовується оператор if з яким більш докладно ми познайомимося на наступній лабораторній роботі.

А обмін значеннями ви маєте написати самі, орієнтуючись на лістинг 3.4 з теоретичної частини.

Лістинг 3.13 – Функція впорядкування значень двох змінних

```
void changeIfNeed(int &a,int &b) {
    if(b < a) {
        //міняємо значення змінних
        . . .
    }
}
```

Тепер без зайвих зусиль можна реалізувати функцію arrange для впорядкування значень трьох змінних, лістинг 3.14.

Перш за все впорядкуємо x1 і x2. Потім x1 і x3. Після цього x1 точно буде мати найменше значення. Залишається впорядкувати x2 і x3.

Лістинг 3.14 – Функція впорядкування значень трьох змінних

```
void arrange(int &x1, int &x2, int &x3) {
    changeIfNeed(  ,   );
    changeIfNeed(  ,   );
    changeIfNeed(  ,   );
}
```

Після цього слід прописати прототипи створених функцій, а далі можна тестувати програму.

3.6 Вміст звіту

Назва роботи (як заголовок розділу 3). Можна скопіювати з методички.

Мета роботи (як підрозділ 3.1). Текст цього підрозділу можна теж скопіювати з методички.

Завдання на роботу (як підрозділ 3.2). Теж можна частково скопіювати з методички.

Результати виконання роботи, як заголовок підрозділу 3.3.

Програмна реалізація завдання, як заголовок пункту 3.3.1. Окрім заголовка в цьому пункті має бути коротенький опис програми і посилання на лістинг 3.1 з кодом програми. Далі має йти сам лістинг.

Контрольний розрахунок для формули як заголовок пункту 3.3.2.

Результати тестування програми, як заголовок пункту 3.3.3. Окрім заголовка в цьому пункті має бути коротенький опис процесу тестування та висновки про результатам тестування і посилання на лістинг 3.2 з результатами тестування (копія тексту консолі). Далі має йти сам лістинг.

Результати виконання програми в режимі налагодження, як заголовок пункту 3.3.4. Короткий опис процесу і рисунок з копією вікна Watches.

Результати дослідження передачі параметрів за значенням, як заголовок пункту 3.3.5. В цьому пункті описати експеримент і зробити висновок.

Результати дослідження передачі параметрів за посиланням, як заголовок пункту 3.3.6. В цьому пункті описати експеримент і зробити висновок.

Висновки, як заголовок підрозділу 3.4. В цьому підрозділі треба коротенько сформулювати що було зроблено та які нові знання та навички отримано.

3.7 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.
- Дати пояснення до звіту.
- Написати функції за вимогою викладача.

4 ЛАБОРАТОРНА РОБОТА 4. ЛОГІЧНИЙ ТИП ДАНИХ І РОЗГАЛУЖЕННЯ У ПРОГРАМАХ

4.1 Мета роботи

– Ознайомитися з операціями над даними логічного типу.
– Навчитися записувати та обчислювати логічні вирази.
– Ознайомитися з операторами **if … else** та **switch**.
– Створити проект, що реалізує алгоритми з розгалуженнями.

4.2 Завдання на лабораторну роботу

– Створити за шаблоном новий проєкт.
– Створити функції, що забезпечать пошук і виведення на консоль коренів квадратного рівняння.
– У функції main() реалізувати тестування усіх гілок алгоритму рішення квадратного рівняння.
– Створити функції для розрахунку стипендії студента за заданим правилом враховуючи статус студента, середній бал і наявність незадовільних оцінок.
– У функції main() реалізувати тестування усіх гілок алгоритму призначення стипендії.

4.3 Теоретична частина

4.3.1 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 x.

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

Зверніть увагу на оперрацію порівняння, яка записується двома символами «дорівнює». Дуже часто студенти роблять помилку використовуючи для порівняння операцію присвоювання. Компілятори С/С++ не помічають такі помилки. Тому будьте уважні!

Таблиця 4.1 - Операції порівняння

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

4.3.1.1 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	Логічні операції	
		&& (і)	|| (або)
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

4.3.1.2 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.

Якщо є потреба обчислювати в першу чергу операції, які мають нижчий пріоритет, то такі операції беруться в дужки.

Таблиця 4.3 – Пріоритети операцій

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

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

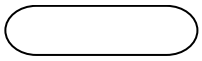
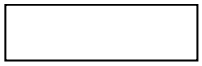
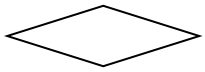
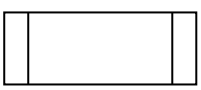
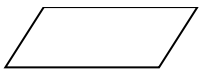
4.3.2 Алгоритми з розгалуженнями

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 ніби план написання програми. Складання таких схем хоча і потребує 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

	Початок и кінець алгоритму
	Обробка інформації, розрахунок за формулою
	Перевірка умови і прийняття рішення
	Зумовлений процес, наприклад, звернення до функції.
	Виведення або введення інформації

Для побудови діаграм існує багато різних засобів. Зокрема ресурс diagrams.net [10].

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 надруковано результати розв'язку квадратного рівняння, у другому – результат лінійного рівняння.

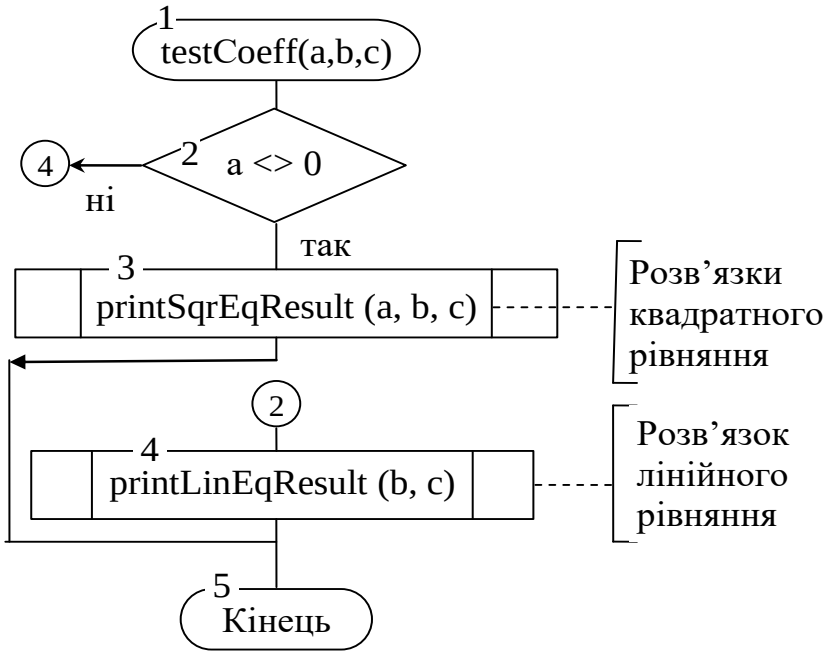


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.1- Алгоритм аналізу коефіцієнтів квадратного рівняння

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 допомогою функції calcLinRoot(), яка використовує формулу $x = -c / b$.

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.

В цьому алгоритмі мають бути наступні кроки:

– обчислення дискримінанта;

– обчислення першої складової кореня $re=–b/(2a)$;

– обчислення другої складової кореня рівняння (корінь квадратний із абсолютної величини дискримінанта $im = \sqrt{|d|}$)

– в залежності від знаку дискримінанта або обчислити дійсні корені (як сума і різниця складових) і надрукувати їх друкувати, або надрукувати комплексі кореня використовуючи обчислені складові .

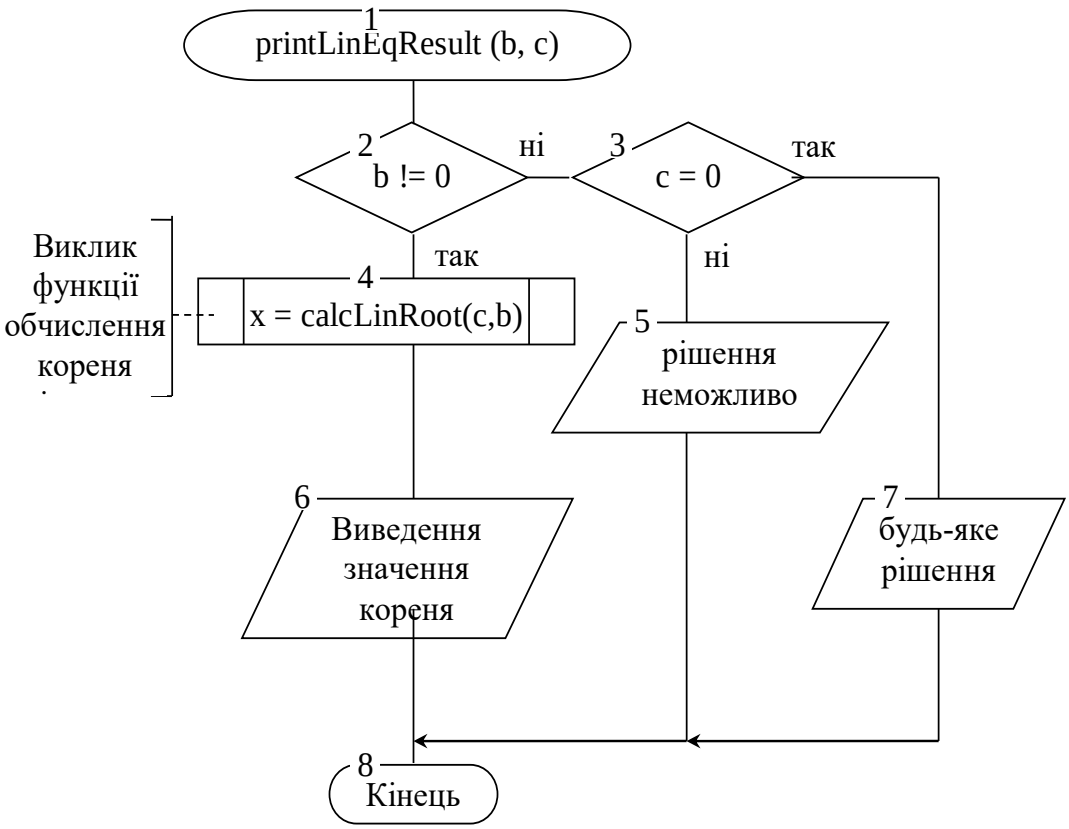


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.2 – Схема алгоритму виводу результатів розв’язку лінійного рівняння

4.3.3 Програмування розгалужень

4.3.3.1 Оператор розгалуження if…else

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

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

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.3– Синтаксис оператору 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 4.1, лістинг 4.1

Лістинг 4.1 – Програмна реалізація аналізу коефіцієнтів рівняння

```
    void testCoeff(double a, double b, double c){
    printf("a = %f,b = %f, c = %f\n",a, b, c);
    if (a==0)
        // Звернення до функції рішення лінійного рівняння
        printLinEqResult(b,c);
    else
        // Звернення до функції рішення квадратного рівняння
        printSqrEqResult(a,b,c);
}
```

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 4.4.

if (<умова>)
<оператор>

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.4 – Скорочена форма оператора if

Оператор **if** може бути вкладений у інший оператор **if**. 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.

Більш зручними і наочними є ланцюжки повних операторів **if**, в яких після кожного слова **else** (за винятком останнього) знову йде повний оператор **if,** рисунок 4.5.

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

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.5 – Приклад ланцюжка повних операторів if

Такі ланцюжки зручно використовувати при вирішенні багатоваріантних задач. Вони досить легко аналізуються, за структурою подібні до розглянутого нижче оператору **switch**, і їх можна застосовувати, на відміну від **switch**, для перевірки будь-яких умов.

4.3.3.2 Умовна операція

Цю операцію називають ще теренарною операцією. Вона виконується над трьома операндами і записується так, як показано на рисунку 4.6.

```
<умова>  ?  < вираз 1 > : < вираз 2 >
```

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.6 – 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. В лістингу 4.2 наведено приклад використання вкладеного теренарного оператора у функції для розрахунку стипендії.

Лістинг 4.2 – Використання теренарного оператора

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

4.3.3.3 Оператор вибору switch

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

Синтаксис оператору **switch** представлено на рисунку 4.7.

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

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.7 – Синтаксис оператора switch

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 **switch** 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, включаючи default, якщо раніше не зустрівся оператор переривання break;

– якщо значення виразу не збігається з жодною із міток, то виконується оператор після слова default, якщо ця частина присутня.

В якості прикладу (лістинг 4.3) розглянемо фрагмент коду, що перераховує значення змінної x в залежності від значення параметру i.

Лістинг 4.3 – Приклад використання операторaswitch

```
int i, x;
. . .
    x=1;
    switch(i) {
        case 1: x=10*i;
        case 2: x+=25*i;
            break;
        case 3: x++;
        default: x+=100;
    }
```

Якщо i = 1, то результатом буде x = 35, тому що спрацюють і перший і другий варіанти.

Якщо i = 2, то результатом буде x = 51, тому що спрацює тільки другий варіант.

Якщо i = 3, то результатом буде x = 102, тому що спрацюють і третій і дефолтний варіанти.

Якщо i = 4, то результатом буде x = 101, тому що спрацює тільки дефолтний варіант.

4.3.4 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 4.8.

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

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.8– Синтаксис оператора goto

Оператор **goto не рекомендують** використовувати, тому що він заплутує програму і робить її малозрозумілою.

4.4 Рекомендації до виконання роботи

4.4.1 Створення проєкту

Створімо за допомогою шаблону новий проєкт. Додаймо директиву підключення бібліотеки математичних функцій.

4.4.1.1 Прототипи функцій

Перед функцією main() можна одразу додати прототипи функцій, які обговорювалися в теоретичній частині. Для прикладу, що розглядається, вони будуть виглядати так, як показано в лістингу 3.6

Лістинг 4.4 – Прототипи функцій проєкту

```
void testCoeff(double a, double b, double c);
void printLinEqResult(double b, double c);
double calcLinRoot(double b, double c);
int calcSqrData(double a, double b, double c,
                  double &re, double &im);
void printSqrEqResult(double a, double b, double c);
```

Функція testCoeff() аналізує перший коефіцієнт рівняння і визначає, чи є рівняння квадратним, чи лінійним Алгоритм роботи цієї функції наведено на рисунку 4.1. Ця функція має викликати або функцію printLinEqResult(), або функцію printSqrEqResult.

Функція printLinEqResult() виводить результат розв'язання лінійного рівняння. В залежності від значень коефіцієнтів тут може бути три варанти відповіді. Схема алгоритму цієї функції наведена на рисунку 4.2. Функція для обчислення значення кореня використовує функцію calcLinRoot().

Функція calcLinRoot() повертає значення кореня, що обраховується за формулою x = -c / b. Звичайно, функція дуже проста, але вона має бути у проекті, бо одне з головних завдань цього проекту – навчитися писати і використовувати функції.

Функція calcSqrData(), обчислює складові коренів рівняння і повертає «знак» дискриминанта (d >= 0). Особливість квадратного рівняння полягає у тому, що корені рівняння можуть бути двох видів – дійсні або комплексні, і форма виведення їх різна. Але у будь якому випадку для формування значень коренів потрібно мати дві складових. Саме ці складові обчислює і повертає через параметри функція calcSqrData(). Перша складова обчислюється за формулою re = -b/(2*a) а друга за формулою im = sqrt(abs(d)/(2*a). Окрім того, функція через return повертає результат обчислення виразу d>=0, що дозволяє перевірити знак дискриминанту d. Схему алгоритму функції calcSqrData() можна знайти у презентації лекції.

Функція printSqrEqResult() виводить результат розв'язання квадратного рівняння. Коли d>=0, то перший корінь рівння є сумою складових x1=re+im, а другий – різницею, x2 = re – im.

Якщо корені комплексні, то виводити результат треба у вигляді двох комплексних чисел. x1=re+(im)*i, x2 = re – (im)* i, де «i» - це умовне позначення кореню із мінус одиниці.

4.4.1.2 Часткова реалізація функції main()

Функція main() має забезпечити виведення на консоль результатів тестування проекту для трьох варіантів лінійного рівняння (звичайне рішення,

нема рішення, безліч рішень) та трьох варіантів квадратного рівняння (дійсні корені, однакові корені, комплексні корені), а також обчислення коренів для рівняння, коефіцієнти якого задає користувач.

Частина реалізації функції main() для прикладу, що розглядається, наведено в лістингу 4.5. Ви маєте її завершити.

Лістинг 4.5 – Часткова реалізація функції main()

```
puts("\n\t Завдання1");
puts("Реалізувати функції розв'язку квадратного рівняння\n"
      "та протестувати можливі варіанти");
double a, b, c;
//Лінійне рівняння. Будь-яке рішення
testCoeff(0,0,0);
//Далі мають бути ще 5 тестів
//. . .
puts("Введіть коефіцієнти: : a = місяць народження,\n"
      " в = день народження, с = рік народження");
cin >> a >> b >> c;
testCoeff(a,b,c);
puts("Введіть рік народження з мінусом");
cin >> c;
testCoeff(a,b,c);
```

Слід зазначити, що після написання цього варіанту головної функції запускати її на виконання нема сенсу, бо ще нема реалізації решти функцій.

Реалізувати ці функції відповідно до прототипів студент має створити самостійно використовуючи схеми алгоритмів та рекомендації з теоретичної частини,

Результат тестування може виглядати приблизно так, як лістинг 4.6 (не всі варіанти перевірено).

Лістинг 4.6 – Результати часткового тестування завдання 1

```
         Завдання1
Реалізувати функції розв'язку квадратного рівняння
та протестувати можливі варіанти

a = 0.000, b = 0.000, c = 0.000
Лінійне рівняння
Коренем може бути будь-яке значення

a = 0.000, b = 0.000, c = 4.000
Лінійне рівняння
Коренів не існує

Введіть коефіцієнти: : a = місяць народження,
 в = день народження, с = рік народження
8 22 1943

a = 8.000, b = 22.000, c = 1943.000
Квадратне рівняння
```

```
Корені x1 = -1.375 + (15.524)i ,x2 = -1.375 - (15.524)i
```

4.5 Завдання для самостійної роботи

Реалізувати функцію для розрахунку стипендії та протестувати її для різних комбінацій параметрів.

Використовувати таке правило призначення стипендії – стипендію отримують студенти бюджетники, як не мають заборгованостей та отримали за результатами сесії середній бал не менше 75. Якщо середній бал вище 95, студент отримує підвищену стипендію. Розмір стипендій можете визначити самостійно.

Слід реалізувати наступні функції:

– функція розрахунку стипендії, яка повертає розмір стипендії (0, розмір звичайної або підвищеної. В якості параметрів приймає ознаку бюджет/контракт, середній бал і кількість заборгованостей;

– функція тестування, яка приймає параметри для розрахунку, виводить їх на консоль, розраховує стипендію і виводить її розмір на консоль;

– доопрацювати функцію main(), в який реалізувати не менше 5 викликів функції тестування (3 варіанти відсутності стипендії і 2 варіанти призначення).

Фрагмент лістингу з результатами виконання цієї частини проєкту може виглядати так, як показано в лістингу 4.6.

Лістинг 4.6 – Результати тестування розрахунку стипендії

```
          Завдання 2
Реалізувати функцію для розрахунку стипендії
та протестувати можливі варіанти
Контракт, бал = 99.00, боргів 0. Стипендія 0.
Бюджет, бал = 69.00, боргів 0. Стипендія 0.
Бюджет, бал = 76.00, боргів 1. Стипендія 0.
Бюджет, бал = 99.00, боргів 0. Стипендія 1500.
Бюджет, бал = 79.00, боргів 0. Стипендія 1000.
Введіть статус студента(бюджет-1, контракт-0),
середній бал сесії та кількість боргів
1 94 0
Бюджет, бал = 94.00, боргів 0. Стипендія 1000.
```

4.6 Вимоги до звіту

Звіт має бути оформлений з урахуванням вимог ДСТУ та відповідно до зразків звітів з попередніх лабораторних робіт.

Не забудьте розмістити у звіті окремим пунктом схему алгоритму знаходження та виведення коренів квадратного рівняння для випадку, коли a>0. За бажанням, замість цієї схеми можна навести схему алгоритму призначення стипендії. Нагадаємо, що для зображення схеми можна скористатися ресурсом [10].

4.7 Контрольні питання

– Запис логічних виразів та 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 функцію обчислення стипендії в залежності від статусу студента (бюджет чи ні), середнього балу, та порогових балів для підвищеної та звичайної стипендії.

5 ЛАБОРАТОРНА РОБОТА 5. ВИКОРИСТАННЯ ЦИКЛІВ WHILE ТА DO…WHILE

5.1 Мета роботи

– Познайомитися з особливостями та принципами використання циклів **while** та **do…while**.

– Познайомитися з алгоритмом пошуку коренів трансцендентного рівняння ітераційним методом.

– Познайомитися з алгоритмом обчислення суми нескінченного ряду з використанням рекурентної формули.

5.2 Завдання на роботу

– Створити за шаблоном новий проєкт.

– Реалізувати пошук кореня трансцендентного рівняння ітераційним методом. Рівняння вибрати з таблиці 5.1 відповідно до номера у списку.

Таблиця 5.1 – Варіанти першого завдання

Варіант	Рівняння	Ітераційна формула	Обмеження
1	$a * x^2 - b * \sin(x) = 0$	$x = \sqrt{\frac{b \cdot \sin(x)}{a}}$	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, 0< b<1, 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	$a \cdot e^{-x} - b \cdot x^2 + c = 0$	$\sqrt{(a * e^{-x} + c)/b}$	a>0, b>0, c>0, x0>0
10	$a \cdot e^{-b \cdot x} - x = 0$	$x = a \cdot e^{-b \cdot x}$	0<a<1, 0<b<1 x0>0

– Реалізувати обчислення суми нескінченного ряду з використанням ітераційної формули. Ряд вибрати з таблиці 5.2 відповідно до номера у списку.

Таблиця 5.2– Завдання на обчислення сум нескінченних рядів

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

5.3 Теоретична частина

5.3.1 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.

5.3.2 Оператор while

Особливість цього 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.

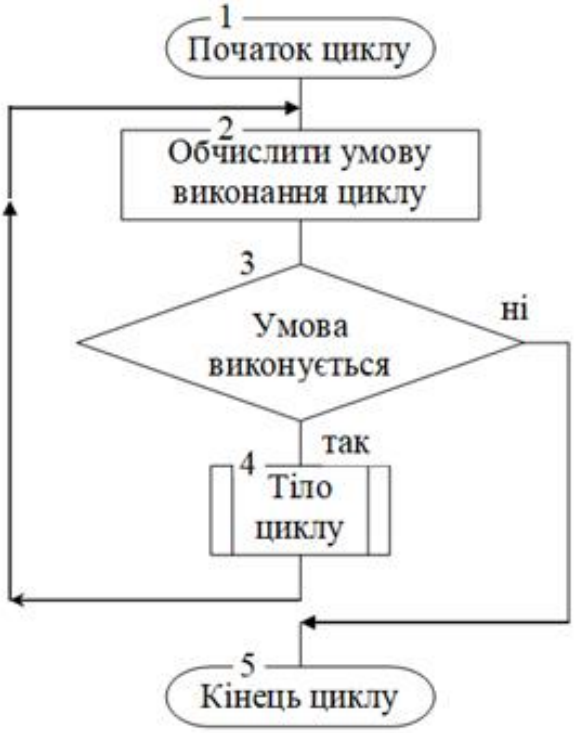


Рисунок 5.2 – Схема алгоритму виконання циклу while

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 в лістингу 5.1

Лістинг 5.1 – Функція підрахунку суми цифр у числі

```
int sumFig(int x){
    int sum=0;
    while(x !=0){
        sum+=x%10;
        x/=10;
    }
    return fabs(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 функції, лістинг 5.2, яка робить те саме, що й попередня.

Такий спосіб написання скорочує код, але робить його важчим для сприйняття, тому не слід зловживати такими можливостями.

Лістинг 5.2 – Варіант функції підрахунку кількості цифр у числі

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

5.3.3 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 {
оператори 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.

Як приклад використання циклу **do…while** розглянемо програму, яка перевіряє пароль. Програма запитує у користувача пароль і якщо введно хибний пароль, запит повторюється.

. Цикл **do…while** тут буде доречним, бо спочатку потрібно ввести пароль, а потім перевірити його правильність. Якщо пароль хибний, то введення паролю слід повторити. Текст програми наведено в лістингу 5.3.

Лістинг 5.3 – Функція перевірки пароля

```
void testParol(int parol){
    int input;
    do{
        puts("/nВведіть пароль");
        cin >> input;
```

```
    }while(parol!=input);
}
```

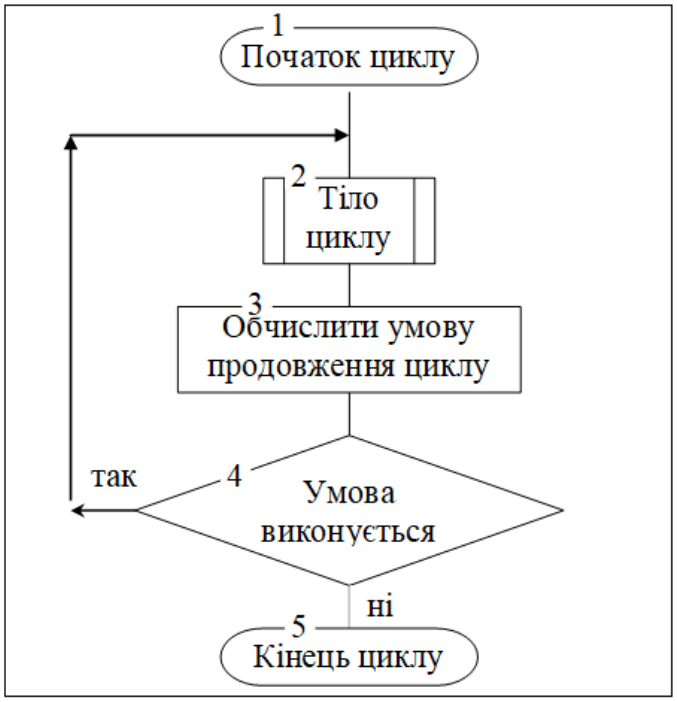


Рисунок 5.4 – Схема алгоритму виконання циклу do…while

5.3.4 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 тільки виконання 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.

5.3.5 Цикл while з умовою всередині тіла циклу

Інколи завершувати цикл **while** зручніше шляхом перевірки умови всередині тіла циклу. В цьому випадку в якості умови для оператора **while** ставимо **true**, тобто створюємо нескінченний цикл. А в тілі циклу перевіряємо умову виходу. Вихід можна реалізувати одним із способів, що розглянуті в попередньому пункті.

5.3.6 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.

Використання ітераційних алгоритмів дозволяє розв'язувати, наприклад, трансцендентні рівняння. У таблиці 5.1 із завданнями на лабораторну роботу наведені приклади рівнянь, які можуть бути вирішені методом ітерацій, і відповідні ітераційні формули. Спосіб отримання ітераційної формули у цих завданнях дуже простий. Рівняння вирішується відносно невідомої змінної, причому в праву частину ітераційної формули входить та ж невідома змінна.

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 за допомогою ітераційних методів полягає в наступному.

Береться якесь наближене до розв'язку значення (початкове наближення) і підставляється до ітераційної формули. Отримане за ітераційною формулою значення розв'язку порівнюється з попереднім. Якщо ці значення істотно відрізняються, 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.

5.3.7 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 алгоритмами, що були розглянуті у попередній роботі, полягає в тому, що обчислення кожного наступного члена ряду проводиться за значенням попереднього.

А відмінність полягає в тім, що обчислені значення членів ряду накопичуються.

Дуже багато стандартних математичних функцій обчислюються саме у такий спосіб. Зокрема, в таблиці 5.2 наведені ряди для обчислення значень таких функцій.

Студент має написати функцію для обчислення суми наведеного ряду, а в функції main() порівняти отримане значення із значенням, що отримано

викликом відповідної стандартної функції.

5.4 Рекомендації до виконання роботи

Створити проєкт для лабораторної роботи за допомогою шаблону.

5.4.1 Реалізація ітераційного алгоритму

5.4.1.1 Формулювання завдання

Завдання, яке ми будемо вирішувати, сформульовано у таблиці 5.3.

Таблиця 5.3 – Завдання на реалізацію ітераційного алгоритму

Варіант	Рівняння	Ітераційна формула	Обмеження
0	$a*e^x - x - b = 0$	$x = a*e^x - b$	$b > a > 0$

5.4.1.2 Аналіз завдання

Для аналізу на рисунку 5.1 побудовано графіки двох функцій змінної x, які відповідають лівій і правій частині ітераційної формули. Значення змінної x у точках перетину цих функцій будуть коренями рівняння, бо в цьому випадку права частина ітераційної формули дорівнює лівій.

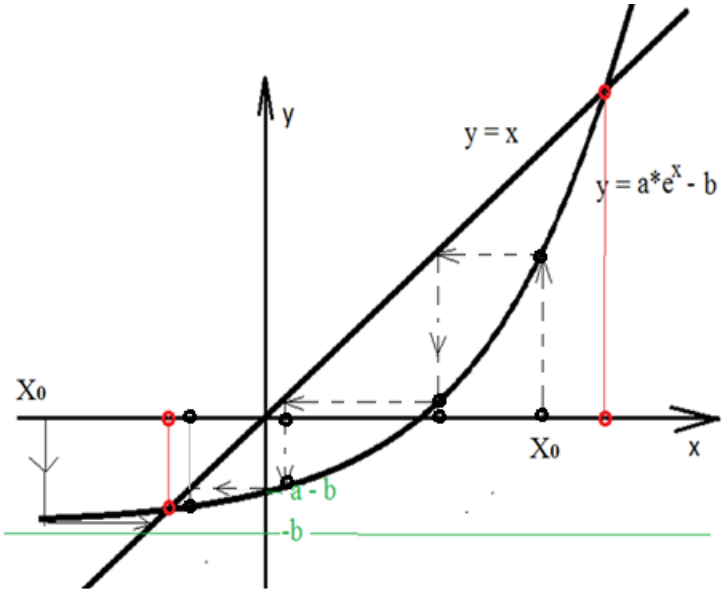


Рисунок 5.1 – Графічна інтерпретація ітераційного методу

Графік функції y = x дозволяє легко переходити на графіку від однієї точки до іншої.

Аналіз показує, що ітераційним методом можна знайти тільки лівий корінь, і тільки за умови, що початкова точка знаходиться будь-де ліворуч від правого кореня. Якщо початкову точку вибрати правіше, то ітераційний процес буде розходитися.

Графік правої частини ітераційної формули перетинає вісь x у точці x = ln(b/a). Оскільки b > a, то координата x точки перетину більша за 0. Тому в якості початкової точки ітераційного процесу можна взяти 0.

5.4.1.3 Реалізація функції пошуку кореня

Код функції наведено в лістингу 5.4. Початкове наближення до кореня задаємо до початку циклу. Цикл створюємо нескінченний. В циклі виводимо послідовні наближення до кореня. Завершується пошук, коли різниця між двома послідовними наближеннями не буде перевищувати мінімально можливе число типу double. Така величезна точність реально не потрібна. Можна було б обмежитися і більшим числом, наприклад 1E-10.

Лістинг 5.4 – Функція пошуку кореня ітераційним методом

```
double findRoot0(double a, double b) {
    //Початкове значення кореня
    double x0 = 0;
    puts("Послідовні наближення:");
    while(true) {
        //Виведення послідовних наближень до кореня
        printf("%f\n", x0);
        //Ітераційна формула
        double x = a*exp(x0) - b;
        //Перевірка умови завершення пошуку
        if(fabs(x - x0)<= DBL_MIN)
            return x;
        //Підготовка до наступної ітерації
        x0 = x;
    }
}
```

5.4.1.4 Реалізація функції main для тестування функції пошуку кореня

Не забудьте включити прототип функції findRoot0.

Функція main() для цієї часини проєкту має забезпечити введення коефіцієнтів рівняння, для якого шукаємо корінь. Крім того потрібно перевіряти значення, що вводить користувач. В разі невиконання умов, яким мають відповідати вхідні дані, програма має повторити запит на їх введення.

Після отримання результату пошуку потрібно зробити перевірку на відповідність рівнянню. Для цього отримане значення треба підставити до рівняння (не в ітераційну формулу!) і обчислити результат. Якщо результат 0, або дуже близький до нього, то це означає, що корінь знайдено правильно.

Відповідний фрагмент функції main() наведено в лістингу 5.5.

Лістинг 5.5 – Частина функції main() для тестування першого завдання

```
puts("\n\t Завдання 1");
puts("Знайти корінь рівняння a*exp(x) - x - b = 0\n"
     "ітераційним методом");
double a, b;
do {
    puts("Введіть коефіцієнти рівняння a, b (b > a > 0)");
    cin >> a >> b;
} while( !(b > a && a > 0));
//Знаходимо корінь
double root = findRoot0(a, b);
printf("Корінь = %1.6f\n",root);
//Перевірка
double test = a*exp(root) - root - b;
if(fabs(test) < 1e-6 )
    puts("Корінь задовольняє рівнянню");
else
    puts("Корінь не задовольняє рівнянню");
```

5.4.2 Реалізація пошуку суми нескінченного ряду

5.4.2.1 Формулювання завдання

В якості прикладу обчислення суми ряду розглянемо алгоритм обчислення синуса деякого числа. Синус можна представити як суму нескінченного ряду, таблиця 5.4, колонка «Ряд».

Таблиця 5.4– Завдання на обчислення сум нескінченних рядів

№	Функція	Ряд	Рекурентна формула
0	sin(x)	$x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots=\sum_{i=1}^{\infty} U_i$	$U_i=-U_{i-1}\cdot\frac{x^2}{(2\cdot i-2)\cdot(2\cdot i-1)}$, для $\lvert x\rvert<2\pi$

Особливість даного ряду полягає в тому, що кожен член ряду, починаючи з другого, може бути знайдений з попереднього за формулою з таблиці 5.4, колонка «Рекурентна формула».

5.4.2.2 Аналіз завдання

В процесі реалізації алгоритму нам знадобиться три змінних, а саме:

– змінна u типу double, яка буде зберігати поточне значення члена ряду;

– змінна sum типу double, яка буде зберігати значення суми членів ряду, які були обчислені на даний момент;

– змінна i типу int, яка буде зберігати поточний номер члена ряду.

Дуже важливо визначитися з початковими значеннями цих змінних. Тут студенти часто роблять помилки. Слід уважно аналізувати ряд і рекурентну

формулу.

В нашому випадку очевидно, перший член ряду і накопичена сума дорівнюють x.

Змінна і за логікою теж має початкове значення 1, але аналіз рекурентної формули показує, що перед розрахунком нового члена ряду цю змінну потрібно збільшувати на 1, бо в формулі використовується номер наступного члена ряду.

Для обчислення нового члена ряду використовується рекурентна формула.

Обчислене значення треба додати до суми членів ряду.

Після цього переходимо до початку циклу обчислень.

Вихід із циклу має відбутися тоді, коли обчислене значення члена ряду стане дуже маленьким, наприклад, не буде більшим 1E-10.

5.4.2.3 Реалізація функція обчислення суми нескінченного ряду

Відповідно до результатів аналізу реалізуймо функцію обчислення синусу як суми нескінченного ряду, лістинг 5.6.

Лістинг 5.6 – Функція обчислення синусу як суми ряду

```
double calcSin(double x){
    //Початкові значення змінних
    double u = x, sum = x;
    int i = 1;
    puts("Послідовні наближення:");
    puts(" i       u              sum");
    while(fabs(u) > 1E-10){
        //Виведення чергового наближення до результату
        printf("%2d   %+1.10f  %+1.10f\n",i ,u, sum);
        //Обчислення змінних
        i++;
        u = -u*x*x/(2*i -1)/(2*i -2);
        sum = sum + u;
    }
    return sum;
}
```

5.4.2.4 Друга частина функції main()

Не забудьте включити прототип функції calcSin.

Функція main() для цієї часини проєкту має забезпечити введення значення змінної, для якої шукаємо синус. Крім того потрібно перевіряти значення, що вводить користувач. В разі невиконання умови, якій має задовольняти змінна, програма має повторити запит на її введення.

Після отримання результату потрібно зробити перевірку на відповідність рівнянню. Для цього треба обчислити синус для введеної змінної за допомогою бібліотечної функції. Якщо результати обчислень співпадають,

або дуже близькі один до ного, то це означає, що функцію реалізовано правильно.

Відповідний фрагмент функції main() наведено в лістингу 5.5.

Лістинг 5.5 – Частина функції main() для тестування першого завдання

```
puts("\n\t Завдання 2");
puts("Обчислити функцію sin(x) як суму ряду\n"
     "ітераційним методом");

double x;
//Цикл введення даних з контролем
do {
    puts("Введіть x (|x| > 2*PI)");
    cin >> x;
} while( fabs(x) > 2*M_PI);
//Знаходимо суму ряду
double sum = calcSin(x);
printf("Сума ряду  = %1.6f\n",sum);
puts("Перевірка");
printf("sin(%1.5f) = %1.6f\n", x, sin(x));
```

5.1 Вимоги до звіту

Звіт має бути оформлений з урахуванням вимог ДСТУ та відповідно до зразків звітів з попередніх лабораторних робіт.

В розділі аналізу не обов’язково наводити рисунок графічної інтерпретації ітераційного пошуку, але для кращого розуміння цього процесу можете його намалювати для себе на папері.

5.2 Контрольні питання

– Опис оператора **while**. Приклад використання.
– Опис оператора **do … while**. Приклад використання.
– Схеми алгоритмів вирішення індивідуальних завдань.
– Пояснення текстів функцій.
– Написати функцію для будь якого варіанту індивідуальних завдань.

6 ЛАБОРАТОРНА РОБОТА 6.
ОБРОБКА ДАНИХ ЗА ДОПОМОГОЮ ЦИКЛУ FOR

6.1 Мета роботи

– Ознайомитися з оператором циклу for та прикладами його використання.

– Ознайомитися з функцією генерації випадкових чисел та прикладами обробки послідовностей випадкових чисел.

– Набути навичок написання програм, що забезпечують обробку даних з використанням циклу for.

6.2 Завдання на роботу

– Створити за шаблоном новий проєкт.

– Реалізувати табулювання значень функції відповідно до варіанту з таблиці 6.1 для певного діапазону значень змінної

Таблиця 6.1 – Завдання на табулювання функції

Варіант	Формула	Змінна
1	$u = 1 + tg(0.1x) + \frac{0.1 \cdot tg(x) + 10}{x}$	*0<x<π/2*
2	$w = 2 \cdot ln(0.5z) + \frac{0.5 \cdot ln(z) + 10}{z}$	*1<z<10*
3	$f = 3 \cdot sin^2(0.5x) - \sqrt{\sin(x)/x}$	*0<x< π*
4	$g = 4 * e^{-0.5t} - \sin(2 * t)/t$	*0<t<π/2*
5	$p = 5 \cdot \sqrt{2\pi} \cdot e^{-0.5t} + t^2$	*0<=t<=2*
6	$q = 6\frac{\ln(2x) + x}{\sqrt{\pi}} + \ln(x+1) + x$	*1<=x<=10*
7	$f = 7\frac{\mathrm{tg}(t/2) + t}{2t}\sqrt{t+\pi} + t$	*0<t< π*
8	$\varphi = 8 \cdot e^x + \frac{x^2 + x + 1}{3x + 2}$	*0<=x<=5*
9	$\mu = 9\ \frac{cos^2(2\pi\omega)}{\omega + 1}\sqrt{\sin(2\pi\omega)}$	*0<=w<0.5*
10	$y = 10x^3 - \sin(x) + \sqrt{\frac{\sin(x)}{x}}$	*0<x< π*

– Реалізувати обробку цілих чисел у відповідності з завданням з таблиці 6.2, обраним відповідно до номера варіанту.

Таблиця 6.2 – Варіанти завдань з обробки цілих чисел

№	Завдання
1	Реалізувати виведення на консоль тризначних цілих чисел, в яких є поспіль тільки дві однакові цифри, і підраховувати їх кількість. Попередньо написати функцію, що визначає, чи задовольняє число заданій умові.
2	Реалізувати виведення на консоль тризначних цілих чисел, в яких крайні цифри співпадають, а середня інша, і підраховувати їх кількість. Попередньо написати функцію, що визначає, чи задовольняє число заданій умові.
3	Реалізувати виведення на консоль тризначних цілих чисел, в яких всі цифри парні, і підраховувати їх кількість. Попередньо написати функцію, що визначає, чи задовольняє число заданій умові.
4	Реалізувати виведення на консоль тризначних цілих чисел, в яких всі цифри непарні, і підраховувати їх кількість. Попередньо написати функцію, що визначає, чи задовольняє число заданій умові.
5	Реалізувати виведення на консоль цілих тризначних чисел в яких кожна наступна цифра більша за попередню. Перед цим написати функцію, що перевіряє, чи відповідає число цій умові.
6	Реалізувати виведення на консоль цілих тризначних чисел в яких кожна наступна цифра менша на 1 за попередню. Перед цим написати функцію, що перевіряє, чи відповідає число цій умові.
7	Реалізувати виведення на консоль цілих тризначних чисел в яких кожна наступна цифра більша на 1 за попередню. Перед цим написати функцію, що перевіряє, чи відповідає число цій умові.
8	Реалізувати виведення на консоль цілих тризначних чисел в яких кожна наступна цифра менша за попередню. Перед цим написати функцію, що перевіряє, чи відповідає число цій умові.
9	Реалізувати виведення на консоль цілих тризначних чисел в яких немає однакових цифр і підраховувати їх кількість. Слід попередньо написати функцію, що визначає, чи є однакові цифри в числі.
10	Реалізувати виведення на консоль тризначних цілих чисел, в яких цифри різні, а їх сума дорівнює 21. Визначити кількість цих чисел. Попередньо написати функцію, що визначає, чи задовольняє число заданій умові.

– Реалізувати обробку послідовності випадкових чисел у відповідності з завданням з таблиці 6.3, обраним відповідно до номера варіанту. Перш за все слід написати функцію, яка послідовно генерує 1000 випадкових чисел в діапазоні від 0 до 100 та формує потрібний результат. Для обробки заборонено використовувати стандартні функції.

Після цього у функції main() в циклі 10 разів звернутися до цієї функції і роздрукувати результат кожного експерименту у вигляді таблиці.

Таблиця 6.3 – Завдання на обробку послідовностей випадкових чисел

Варіант	Характеристика послідовності, що має бути обчислена
1	Максимальне число
2	Мінімальне число
3	Середнє арифметичне значення
4	Середнє арифметичне значення квадратів чисел
5	Кількість кратних 5
6	Кількість парних чисел
7	Кількість непарних чисел
8	Різниця між максимальним та мінімальним числом (розмах)
9	Кількість парних чисел
10	Кількість непарних чисел

6.3 Теоретична частина

6.3.1 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 так, як показано на рисунку 6.1.

```
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 продовження 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.

6.3.1.1 Алгоритм виконання циклу for

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.

6.3.1.2 Обчислення числа Фібоначчі у циклі for

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 …

У функції, яку наведено в лістингу 6.1 послідовно обчислюються числа Фібоначчі починаючи з другого і до заданого, з номером n.

Лістинг 6.1 – Функція обчислення числа Фібоначчі за заданим номером

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

6.3.1.3 Обчислення факторіалу у циклі for

У лістингу 6.2 цикл for використовується для підрахунку факторіала числа n.

Нагадаємо, що факторіал числа n дорівнює добутку цідих чисел від 1 до n. Факторіал нуля вважається рівним 1. Слід розуміти що факторіал зростає дуже швидко, тому для результату обрано тип unsigned long ling int.

Лістинг 6.2 – Функція обчислення факторіала числа

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

6.3.2 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 так, як наведено в лістингу 6.3.

Лістинг 6.3 – Другий варіант обчислення факторіалу

```
unsigned long long fact2(int n) {
    unsigned long long f=1;
    int 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, як наведено у лістингу 6.4.

Лістинг 6.4 – Функція підрахунку суми цифр у числі

```
int calcFig(unsigned int n){
    int x = n, sum=0;
    for( ; x != 0; ){
        sum += x%10;
        x /= 10;
    }
    return sum;
}                                   }
```

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, лістинг 6.5.

Лістинг 6.5 – Ще один спосіб обчислення факторіалу

```
unsigned long long fact3(int n) {
    unsigned long long f = 1;
    for( int i = 1; i <= n; f *= i, i++ );
    return f;
}
```

А так, як показано в лістингу 6.6, може виглядати функція обчислення числа Фібоначчі.

Лістинг 6.6 – Ще один спосіб обчислення числа Фібоначчі

```
int fibo2(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;
}
```

6.3.3 Випадкові числа

Для формування випадкових чисел в мові С використовується функція rand(), 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.

Щоб використовувати згадані вище функції та константи, необхідно підключити заголовний файл <stdlib.h>.

Функцію srand() 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 rand() 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 srand() може виглядати так:

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

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

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

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

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

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

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

6.4 Рекомендації до виконання роботи

Створімо проєкт для лабораторної роботи за допомогою шаблону. Доцільно зразу ж додати підключення заголовних файлів <cmath>, <stdlib.h>, <time.h>.

6.4.1 Табулювання функцій

Суть завдання полягає в тому, що необхідно надрукувати таблицю значень функції для послідовності аргументів, які змінюються з деяким кроком. Цю задачу можна вирішувати за допомогою циклу for.

6.4.1.1 Завдання на табулювання функції

В якості прикладу розглянемо табулювання функції, яку наведено в таблиці 6.4.

Таблиця 6.4 – Завдання на табулювання функції

Варіант	Формула	Змінна
0	$y = 2e^{(-0.1x)} * \sin(0.5x)$	$0<=x<=2\pi$

6.4.1.2 Аналіз завдання

Для того, щоб не перевантажувати функцію main() зайвими деталями, розділимо завдання на три складові частини і реалізуємо ці складові у вигляді окремих функцій. Реалізація складових частин будь-якої задачі у вигляді окремих функцій є дуже гарною і розповсюдженою практикою. Ц робить програму зрозумілою, надійною, спрощує налагодження, модифікацію та повторне використання коду.

У своїх проєктах студенти обов’язково мають дотримуватися такої практики і реалізовувати завдання за допомогою таких функцій:

– функція для розрахунків за формулою;
– функція введення і контролю даних для формування таблиці;
– функція виведення таблиці на консоль.

Відповідно до цього створімо прототипи цих функцій, лістинг 6.7.

Лістинг 6.7 – Прототипи функцій для першого завдання

```
double funcForTab(double x);
void inptTabParm(double &min, double &max, double &step);
void printTable(double min, double max, double step);
```

6.4.1.3 Реалізація функції для розрахунків за формулою

Реалізацію функції наведено в лістингу 6.8.

Лістинг 6.8 – Реалізація функції для розрахунків за формулою

```
//Функція розрахунків за формулою 2(e^(-0.1x))sin(0.5x)
double funcForTab(double x) {
    double y = 2*exp(-0.1*x)* sin(0.5*x);
    return y;
}
```

6.4.1.4 Реалізація функції введення і контролю даних

Ця функція, лістинг 6.9, забезпечує введення початкового та кінцевого значень аргументу і кроку його зміни. Початкове значення має бути меншим за конечне, а різниця між ними має бути в декілька разів більша кроку. Крім того вони мають задовольняти умовам таблиці 6.4

Лістинг 6.9 – Функція введення і контролю даних

```
//Функція ведення та контролю параметрів таблиці
void inptTabParm(double &min, double &max, double &step) {
    while(true) {
        puts("Введіть початкове, кінцеве значення\n"
             " та крок зміни аргументу функції");
        cin >> min >> max >> step;
        if(min>=0 && max <= 2*M_PI && min<max
                && (max-min) > 2*step) break;
    }
}
```

6.4.1.5 Реалізація функції виведення таблиці значень на консоль

Це завдання легко реалізується за допомогою циклу for. В якості параметра циклу буде змінна x, яка є аргументом функції.

Код функції наведено в лістингу 6.10.

Лістинг 6.10 – функції виведення таблиці значень функції на консоль

```
//Функція виведення на консол таблиці значень функції
void printTable(double min, double max, double step){
    //Заголовок таблиці
    puts("  x       y ");
    //Виведення таблиці
    for(double x = min; x <= max; x+=step) {
        double y = funcForTab(x);
        printf("%6.3f %8.3f\n",x, y);
    }
}
```

6.4.1.6 Реалізація першого завдання в функції main()

Реалізацію цієї частини функції main() наведено в лістингу 6.11. Як бачимо, код не засмічений деталями і чітко фіксує кроки виконання завдання.

Лістинг 6.11 – Перша частина функції main()

```
puts("\n\t Завдання 1");
puts("Надрукувати таблицю значень заданої функції");
double min, max, step;
//Введення та контроль вхідних даних
inptTabParm(min, max, step);
puts("Таблиця значень функції\n"
     "y = 2(e^(-0.1x))sin(0.5x)");
printTable(min, max, step);
```

6.4.2 Обробка послідовностей цілих чисел

Завдання цього етапу лабораторної роботи полягають в послідовному перегляді додатних цілих чисел в діапазоні від 100 до 999 та перевірці кожного з них на відповідність певним ознакам. Для перевірки, чи має число задані ознаки, створюється окрема функція. Числа, що відповідають цим ознакам виводяться на консоль. Крім того підраховується кількість цих чисел.

6.4.2.1 Формулювання завдання.

В якості прикладу реалізуймо завдання з таблиці 6.5.

Таблиці 6.5 – Завдання на обробку послідовності цілих чисел

№	Завдання
0	У послідовності трицифрових парних цілих чисел знайти такі, в яких сусідні цифри мають різну парність. Попередньо написати функцію, що визначає, чи задовольняє число такій умові.

6.4.2.2 Аналіз завдання

Для вирішення цієї задачі спочатку напишемо допоміжну функцію, що з'ясовує, чи сусідні цифри мають різну парність.

Після цього можна написати функцію, яка в циклі for буде перевіряти на задану ознаку усі числа трицифрові числа і друкувати ті з них, що задовольняють умові. Крім того ця функція буде повертати кількість знайдених чисел. А функція main() буде просто викликати цю функцію.

Прототипи функцій, які будемо реалізовувати, наведені у лістингу 6.12.

Лістинг 6.12 – Прототипи функцій реалізації другого завдання

```
int testDigits(int n);
int selectAndPrintNumber();
```

6.4.2.3 Розробка функції тестування числа

Функцію, що з'ясовує, чи сусідні цифри числа мають різну парність, назвемо testDigits. Якщо число, яке перевіряє функція, задовольняє умові, функція має повертати true, у іншому випадку має повертати false.

Текст функції наведено в лістингу 6.13.

Лістинг 6.13 – Код допоміжної функції

```
//Функція, що визначає, чи мають сусідні числа різну парність
int testDigits(int x) {
    int d3 = x%10, d2 = x/10%10, d1 = x/100;
    return d1%2 == d3%2 && d1%2 != d2%2;
}
```

6.4.2.4 Реалізація функції обробки послідовності чисел

Текст функції наведено в лістингу 6.14.

Для формування послідовності чисел використовуємо цикл for. Кількість чисел, що задовольняють умові, накопичується в лічильнику cnt.

Між числами в рядку вставляється пробіл, але якщо кількість чисел кратна 15, то виводиться символ переходу на новий рядок.

Лістинг 6.14 – Функція формування і обробки послідовності чисел

```
//Функція друкує рядки по 15 чисел, що задовольняють умові,
//і повертає їх кількість
int selectAndPrintNumber() {
    int cnt = 0;
    for(int x = 100; x <= 999; x++) {
        if(testDigits(x)) {
            cnt++;
            printf("%d%c", x, cnt%15==0 ? '\n' : ' ');
        }
    }
    return cnt;
}
```

6.4.2.5 Реалізація другої частини у функції main()

Маючи реалізовані функції можемо перейти до вирішення другого завдання у фунції main(). В лістингу 6.15 наведено текст відповідної частини функції main()

Лістинг 6.15 – Реалізація другої частини функції main()

```
puts("\n\t Завдання 2");
puts("Знайти тризначні цілі числа,\n"
     "в яких сусідні цифри мають різну парність");
int n = selectAndPrintNumber();
printf("\nУсього знайдено %d чисел\n", n);
```

6.4.3 Обробка послідовності випадкових чисел

Нагадаємо, що на цьому етапі реалізації проєкту, перш за все слід написати функцію, яка послідовно генерує 1000 випадкових чисел в діапазоні від 0 до 100 та формує потрібний результат.

Після цього у функції main() в циклі треба 10 разів звернутися до цієї функції і роздрукувати результат кожного експерименту у вигляді таблиці.

6.4.3.1 Формулювання завдання.

В якості прикладу реалізуймо завдання з таблиці 6.6.

Таблиці 6.6 – Завдання на обробку послідовності цілих чисел

№	Завдання
0	Знайти найбільшу різницю між наступним і попереднім числом у послідовності випадкових чисел

6.4.3.2 Розробка функції, що обчислює задану характеристику

Для отримання чисел випадкової послідовності будемо використовувати функцію rand() із урахуванням рекомендацій пункту 6.3.3.

Текст функції наведено в лістингу 6.16.

Лістинг 6.16 – Функція обробки випадкової послідовності

```
//Функція, що знаходить найбільшу різницю між наступним
//і попереднім числом у послідовності випадкових чисел
int maxDif() {
    int pred = rand()%101;
    int max = 0;
    for(int i = 1; i < 1000; i++) {
        int x = rand()%101;
        int dif = x - pred;
        if(dif > max) max = dif;
        pred = x;
    }
    return max;
}
```

В цій функції спочатку визначаємо змінну pred для збереження попереднього числа та ініціалізуємо її, згенерувавши перше випадкове число.

Визначаємо також змінну max для збереження найбільшої різниці між наступним і попереднім та присвоїмо їй значення 0.

Після цього в циклі генеруємо наступні випадкові числа, визначаємо різницю і, якщо вона більша за max, записуємо її в max. Таким чином ми визначимо найбільше значення.

Після аналізу поточного числа робимо його попереднім і продовжуємо цикл обробки.

6.4.3.3 Реалізація останньої частини функції main

В цій частині функції main() перш за все слід звернутися до функції srand для ініціалізації генератора випадкових чисел

Після цього достатньо 10 разів звернутися до попередньої функції і надрукувати результат. Код цієї частини функції main() наведено в лістингу 6.17.

Лістинг 6.17 – Реалізація третьої частини функції main()

```
puts("\n\t Завдання 3");
puts("Провести 10 експериментів з визначення\n"
     "найбільшої різниці між наступним і попереднім\n"
     "числом у послідовності випадкових чисел");
srand(time(NULL) | clock());
for(int i = 0; i < 10; i++) {
    printf("Найбільша різниця між наступним і "
           "попереднім %d\n", maxDif());
}
```

6.5 Вимоги до звіту

Звіт має бути оформлений з урахуванням вимог ДСТУ та відповідно до зразків звітів з попередніх лабораторних робіт.

6.6 Контрольні питання

– Опис оператора for. Приклад використання.

– Приклад використання оператора for з операцією кома.

– Особливості використання функції qrand().

– Написати функцію для обробки випадкових чисел за вказівкою викладача з використанням циклу for.

– Написати функцію або програму відповідно до одного із варіантів індивідуальних завдань до лабораторної роботи.

7 ЛАБОРАТОРНА РОБОТА 7.
ОСНОВИ РОБОТИ З ВКАЗІВНИКАМИ

7.1 Мета роботи

– Ознайомитися з поняттям вказівник.
– Навчитися оголошувати та ініціалізувати вказівники.
– Ознайомитися з функціями виділення пам’яті.
– Отримати практичні навички обробки даних за допомогою вказівників

7.2 Завдання на роботу

– Заповнити ділянку пам’яті розміром 100 + дата народження випадковими тризначними цілими числами та надрукувати ці числа.

– Реалізувати універсальну функцію обробки ділянки пам’яті з випадковими числами **до якої передається вказівник на функцію тестування** чисел у відповідності з завданням з таблиці 6.2 (минула лабораторна робота). Функція має повертати вказівник на нову ділянку пам’яті з вибраними числами та їх кількість. У функції main() надрукувати послідовність чисел отриману після обробки послідовності, яку було перед цим згенеровано.

– 10 разів наповнити ділянку пам’яті випадковими числами і для кожної вибірки чисел обчислити і надрукувати характеристику у відповідності з завданням з таблиці 6.3 (минула лабораторна робота).

– Реалізувати універсальну програму виведення таблиці значень функції, **до якої передається вказівник на функцію розрахунку за формулою** у відповідності з завданням з таблиці 6.1 (минула лабораторна робота).

7.3 Теоретична частина

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 всі мови програмування надають можливість використовувати вказівники, але у таких популярних мовах програмування як С та Паскаль робота з вказівниками можлива. Більш того, можна сказати, що вказівники – це візитна картка мови С.

Вважається, що використання вказівників дає змогу скоротити текст програми та підвищити її ефективність. Але тут є і зворотна сторона. Вказівники – це не зовсім прості у використанні засоби програмування і часто стають джерелом помилок, виявити які у програмі дуже складно. Тому в деяких мовах прикладного програмування (наприклад, Java) принципово відмовилися від вказівників.

7.3.1 Оголошення та ініціалізація вказівників

Оголошення вказівника має вигляд, представлений на рисунку 7.1.

<тип даних> *<ім’я вказівника> ;

Рисунок 7.1 – Синтаксис оголошення вказівника

У цьому оголошенні тип даних визначає тип елемента програми, адресу якого може зберігати даний вказівник. Це може бути будь який допустимий у мові тип, простий або складений

Ім’я вказівника – це ідентифікатор написаний за правилами запису імен у мові С.

* – це ознака того, що наступне ім’я є вказівником.

Приклад оголошення вказівника 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 вказівника, тому в разі оголошення декількох вказівників символ * треба ставити перед кожним іменем. Нижче наведено приклад оголошення двох вказівників pw та pz одночасно із оголошенням простої змінної pv:

```
int  *pw, *pz, pv ;
```

Під час оголошення вказівника його можна ініціалізувати константою цілого типу, але цю константу слід явно привести до типу вказівника. У прикладі наведеному нижче вказівнику pd на дані типу double присвоюється значення 1024 (нагадаємо, що значення вказівника – це адреса даних):

```
double  *pd = (double*)1024;
```

Зверніть увагу, у конструкції, що використовується для приведення до типу вказівника символ * відноситься вже до назви типу даних.

Вказівник 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 вказівник pz, який проініціалізований адресою змінної z. Таким чином вказівник pz містить адресу змінної z.

Якщо вказівник 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 вказівник, йому присвоюють значення NULL. Вважається, що вказівник, значенням якого є NULL, ні на що не посилається і його називають порожнім вказівником. Нижче наведено

приклад такої ініціалізації:

```
int  *pw = NULL ;
```

7.3.2 Звернення до даних через вказівники

Якщо відома адреса 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 вказівника. Звертання до даних через вказівник можна використовувати всюди, де можна звертатися до даних через ім'я змінної.

Нехай оголошено зміну z і проініціалізовано вказівник 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 вказівниками у програмах на мові С слід пам'ятати, що ніякого контролю за значеннями вказівників, 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 програми.

7.3.3 Використання кваліфікатора const для вказівників

Вказівники 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 вказівника:

```
const int z = 100;
const int *ptc = &z;
```

Зверніть увагу, ініціалізувати вказівник на константні дані можна тільки адресою константи.

Вказівники на константні дані часто використовуються в оголошеннях параметрів функцій.

Другий варіант використання кваліфікатора const передбачає незмінність самого вказівника. Для реалізації цього варіанту слово const має бути записано перед іменем вказівника:

```
int  *const cpt = &z;
```

Оголошений таким чином вказівник cpt не може змінювати свого значення, а змінна z на яку він посилається змінюватися може. Такі вказівники називають константними.

Третій варіант використання кваліфікатора const передбачає незмінність як самого вказівника, так і даних, на які він посилається. Для реалізації цього варіанту слово const має бути записано двічі, перед типом та перед іменем вказівника:

```
const int z = 100;
const int *const ptc = &z;
```

7.3.4 Адресна арифметика

Над вказівниками можливі такі операції:

– присвоєння;
– порівняння;
– збільшення/зменшення;
– віднімання.

За допомогою операції присвоєння вказівнику можна присвоїти значення адресної константи, адресу якоїсь змінної або результат обчислення виразу, що знаходиться праворуч від знаку присвоєння. Необхідною умовою операції присвоєння для вказівників є однаковість базових типів вказівника і значення, що йому присвоюється.

Для порівняння вказівників можна використовувати звичайні операції порівняння. Найчастіше з цих операцій використовуються операції == та !=.

До значення вказівників можна додавати (або віднімати) цілі числа. При цьому значення вказівника 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 вказівника ptn буде на 20 більшим, ніж значення вказівника 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 вказівника, наприклад, на величину k, формує адресу елементу даних, розташованого на k елементів правіше, або лівіше у разі операції віднімання.

Для зміни значень вказівників можна також застосовувати операції інкременту та декременту. Операція інкременту зміщує вказівник до наступного елементу, а декременту – до попереднього. Найчастіше такі операції використовуються під час роботи з масивами.

Операція віднімання, що виконується над двома вказівниками повертає кількість елементів базового, типу що може розміститися між адресами, на які вказують перший та другий операнди операції.

7.3.5 Нетипізовані вказівники

У мові С можна використовувати вказівники, які не пов’язані з конкретним типом і сумісні з вказівниками на будь які типи даних. Використання безтипових вказівників дозволяє підвищувати ефективність програм. Безтипові вказівники оголошуються із ключовим словом void:

```
void *pz;
```

Хоча значення без типових вказівників 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, лістинг 7.1, який дозволяє отримати доступ до двох половин числа типу int (int бере пам'яті удвічі більше за short).

Лістинг 7.1 – Приклад отримання доступу до частин числа типу int

```
int number = 0x12AB34DC;
void *p= &number;
short  part1, part2;
part1=*(short*)p;
part2=*((short*)p+1);
```

У цьому прикладі ми розрізали число number типу int на числа part1 та part2 типу short.

7.3.6 Вказівники, як параметри функцій

Вказівник також є елементом даних, тому його можна передавати до функцій як параметр і повертати як результат роботи функції.

7.3.6.1 Передача вказівників за значенням

Розглянемо функцію, яка виконує обмін значень двох змінних. Раніше ми вирішували це завдання використовуючи посилання. Але замість посилань можна передати вказівники на змінні і вирішити задачу, лістинг 7.2.

Лістинг 7.2 – Функція обміну значень двох зміних

```
void xchng(int *a, int *b){
    int c = *a;
    *a = *b;
    *b = c;
}
```

Для тестування такої функції можна скористатися такою функцією main(), лістинг 7.3.

Лістинг 7.3 – Функція для тестування функції обміну даними

```
int main() {
    int x =10, y = 20;
    xchng (&x, &y);
```

```
    cout << x << y << endl;
}
```

Аналізуючи лістинги 7.2 та 7.3 і порівнявши їх з лістингами 3.4 та 3.5 можна зробити висновок, що краще передавати до функцій посилання на змінні, а не вказівники на ці змінні.

7.3.6.2 Передача вказівників за посиланням

Передача вказівника у функцію за посиланням дозволяє функції поміняти значення цього вказівника.

Розглянемо приклад функції обміну значень вказівників:

```
void ptrchng(int *&a, int *&b) {
     int *c = a;
     a = b;
     b = c;
}
```

Для тестування цієї функції можна використати код, що наведено нижче:

```
int main() {
    int x =10, y = 20;
    int *px = &x, *py = &y;
    ptrchng (px, py);
    cout << *px << *py << endl;
}
```

7.3.6.3 Вказівник як результат функції

В якості прикладу розглянемо функцію, яка повертає вказівник на більшу з двох змінних:

```
int* max(int* x1, int* x2) {
  if(*x1 > *x2) return x1;
  return x2;
}
```

Тестувати цю функції можна за допомогою такого коду:

```
int main() {
  int z =5, v =7;
  int *m = max(&z, &v);
  cout << *m << endl;
}
```

Використовуючи вказівники як результати роботи функцій слід пам’ятати, що локальні змінні, які оголошено в межах функції існують тільки до завершення роботи функції. Тому повертати посилання на такі змінні нема сенсу.

7.3.7 Вказівники на функції

Оскільки функція – це ділянка пам’яті, де розташовано її код, і ця ділянка має свою адресу, то можна сформувати і вказівник на цю ділянку пам’яті. Фактично, ім’я функції і є таким вказівником.

Але для того, щоб оголосити вказівник на функцію і потім його використовувати ми маємо визначити тип цього вказівника, тобто визначити тип функції.

7.3.7.1 Тип функції

Для того, щоб зрозуміти, що таке тип функції, порівняємо два вирази, що оперують дійсними числами:

```
y = a*sin(w*x) + sqrt(a);
z = b*x – log(x) +d;
```

Як бачимо, вирази суттєво відрізняються.

Для обчислення цих виразів можна написати дві функції – calcY, calcZ.

Напишемо прототипи цих функцій, не використовуючи імена змінних:

```
double calcY(double, double, double);
double calcZ(double, double, double);
```

Виявляється, що незважаючи на суттєву різницю виразів, прототипи функцій дуже схожі. Різниця тільки в назві. Можна вважати, а так воно і є, що ці функції одного типу. Тобто тип функції визначає, що функція повертає, скільки параметрів вона приймає і якого вони типу.

Для оголошення типу функції використовується директива #define. Для оголошених нам функцій визначення типу буде таким:

```
#typedef double TypeFunc(double, double,double);
```

Оголошення типу, як бачимо, схоже на прототип функції. Тільки замість назви функції ми пишемо назву типу. І все це робиться у директиві #define.

Маючи тип функції можна оголосити змінну, яка буде вказівником на функцію:

```
TypeFunk *pf;
```

Слід зауважити, що є і інші способи оголошення типу функції, але ті способи важче сприймаються і запам’ятовуються. Докладно познайомитися із способами оголошень вказівників на функції можна у підручнику[1], сторінка 212.

7.3.7.2 Використання вказівників на функції

Вказівники на функції використовуються для вирішення різних завдань. Докладно ці питання розглядаються у розділі 11.9 підручника, сторінка 212.

Ми ж розглянемо тут тільки один приклад.

Хай нам потрібно отримати таблиці значень функцій calcY та calcZ, для

яких у попередньому підпункті були оголошені прототипи та тип.

З огляду на той факт, що ці функції одного типу, ми можемо написати одну узагальнену програму роздруківки таблиці значень для цих функцій і будь-якої іншої функції цього типу. Ім’я потрібної функції будемо передавати до цієї універсальної програми через параметр.

Узагальнена програма (теж функція) може виглядати так, як показано в лістингу 7.4.

Лістинг 7.4 – Узагальнена програма виведення таблиці значень функції

```
void printFunc(double x0, double xMax, double step,
               double parm1, double parm2, FuncType *pf){
    for(double x = x0; x < xMax; x+=step)
        printf("%10.3f %10.3f\n", x, pf(parm1, parm2, x));
}
```

Далі ми реалізуємо функції calcY та calcZ, лістинг 7.5.

Лістинг 7.5 – Реалізації функцій обчислення за формулами

```
double calcY(double a, double w, double x){
return a*sin(w*x) + sqrt(a);
}

double calcZ(double b, double d, double x){
    return b*x - log(x) + d;
}
```

Після цього можна друкувати таблички значень цих функцій, використовуючи одну і ту ж саму програму роздруківки, лістинг 7.6.

Лістинг 7.6 – Приклади звернень до узагальненої функції

```
int main()
{
    printFunc(0,1,0.1, 2.5, 5, calcY);
    printFunc(1,2,0.2, 3, 7, calcZ);
    return 0;
}
```

7.3.8 Операції з пам’яттю

Поняття вказівника і пам’яті комп’ютера пов’язані між собою.

Розрізняють статичну, динамічну і стек, який теж є частиною пам’яті.

Статична пам’ять виділяється на етапі компіляції програми і є складовою частиною програми.

Стек використовується для обміну інформацією між функціями та забезпечення їх коректної роботи.

Динамічна пам’ять виділяється під час виконання програми з так званої «купи» (heap).

Робота з динамічною пам’яттю має наступні складові:

– отримання ділянки пам’яті в користування;
– використання цієї ділянки для обробки даних;
– звільнення цієї ділянки пам’яті.

7.3.8.1 Отримання доступу до ділянки динамічної пам’яті

Перший спосіб вирішення цього завдання – використання функції malloc, рисунок 7.2.

```
void   *malloc(<розмір ділянки пам’яті в байтах> ;
```

Рисунок 7.2 – Функція malloc

Функція повертає вказівник на перший байт виділеної ділянки пам'яті. Розмір ділянки визначається аргументом функції, але корегується таким чином, щоб там вмістилася ціла кількість даних будь якого базового типу.

Якщо пам’ять виділити не вдалось, функція повертає NULL.

Для використання функції необхідна підключити заголовний файл <stdlib.h>. Нижче наведено приклад виклику цієї функції:

```
int *p = (int*)malloc(20*sizeof(int));
```

Другий спосіб, більш сучасний, полягаю у використанні службового слова new. Перевага цього способу полягає в тім, що одразу визначається тип даних, що будуть зберігатися в цій ділянці, і розмір задається не в байтах, а в кількості даних заданого типу.

Якщо потрібна ділянка для одного елемента даних, то розмір взагалі не вказується:

```
int *a = new int;
```

Якщо потрібно виділити пам'ять для одного елемента разом з його ініціалізацією, то синтаксис буде такий:

```
int *b = new int(5);
```

Виділення пам’яті для послідовності 5 змінних типу int буде виглядати так:

```
int *c = new int[5];
```

Зверніть увагу на різницю в дужках в двох попередніх виразах. На жаль, схожість виразів є причиною прикрих помилок.

Використання ділянок пам’яті

Наявність вказівника на початок ділянки пам’яті та адресна арифметика дозволяють дуже просто обробляти дані в цій ділянці.

В якості прикладу розглянемо функцію, яка наповнює ділянку пам’яті випадковими числами, лістинг 7.7.

До функції, в якості параметрів передається вказівник data на ділянку пам’яті та розмір ділянки в числах int.

Для переміщення по ділянці і запису чисел використовується вказівник p.

Лістинг 7.7 – Функція наповнення числами ділянки пам’яті

```
//Функці, що наповнює ділянку пам'яті випадковими числами
void fillWithRnd(int *data, int size) {
    //Початок області пам'яті за межами ділянки
    int *end = data + size;
    //Цикл послідовного занасення чисел до ділянки
    for(int *p = data; p < end; p++) {
        //Формування випадкового числа в межах від 100 до 999
        *p = 100 + rand()%900;
    }
}
```

7.3.8.2 Звільнення пам’яті

Особливість C/C++ полягає, зокрема, в тім що ці мови не забезпечують автоматичного збору сміття. Термін «сміття» тут означає ділянки динамічної пам’яті, які програма вже не використовує. Тому обов’язок С-програміста слідкувати за використанням пам’яті і звільняти ділянки, які не потрібні. На жаль, не завжди це робиться (людський фактор!), внаслідок чого обсяг доступної пам’яті може поступово зменшуватися. Цей процес зветься «memory leak» – витікання пам’яті і він може призвести, зрештою, до руйнування програми.

Якщо пам’ять виділялася функцією malloc, то для її звільнення потрібно використовувати функцію free, до якої в якості параметра передається вказівник, що був повернутий функцією malloc:

```
free (ptr);
```

Якщо пам’ять виділялася через new то в цьому випадку слід використовувати delete[] або delete, якщо пам’ять виділялась тільки для одного елемента даних, наприклад:

```
delete a; delete[]data;
```

7.3.8.3 Стандартні функції для роботи з ділянками оперативної пам'яті

Декілька таких функцій знаходиться в бібліотеці, на яку посилається заголовний файл <string.h> . На даному етапі для нас можуть бути цікаві дві з них.

Функція memcpy переписує задану кількість байтів з однієї ділянки пам’яті у другу, але ділянки не повинні перетинатись.

Функція memmove робить те саме, але дозволяє, щоб ділянки перетинатись.

Обидві функції схожу сигнатуру, рисунок 7.3 і повертають безтиповий вказівник на перший байт ділянки, куди були скопійовані дані.

На рисунку 7. 3 <куди> та <звідки> – це вказівники на перші байти ділянок пам'яті.

```
void   *memcpy(<куди> , <звідки>, <сільки>;
```

Рисунок 7.3 – Функція memcpy

7.4 Рекомендації до виконання роботи

Створімо проєкт для лабораторної роботи за допомогою шаблону. Доцільно зразу ж додати підключення заголовних файлів <cmath>, <time.h>.

7.4.1 Реалізація завдання 1

На цьому етапі потрібно «Заповнити ділянку пам'яті розміром 100 + дата народження випадковими тризначними цілими числами та надрукувати ці числа».

7.4.1.1 Аналіз завдання

Розмір ділянки пам'яті має дорівнювати 100 + 22 чисел типу int.

Для реалізації потрібно дві функції.

Перша – для генерації чисел та заповнення ними ділянки пам'яті. Така функція вже розглядалась в якості прикладу, лістинг 7.7.

Друга функція потрібна для роздруківки цих чисел.

Напишемо прототипи цих функцій, лістинг 7.8. До кожної з них в якості параметрів передаються посилання на початок ділянки пам'яті та кількість чисел в цій ділянці.

Лістинг 7.8 – Прототипи функцій для першого завдання

```
void fillWithRnd(int *data, int size);
void printData(int *data, int size);
```

7.4.1.2 Реалізація функції виведення на консоль чисел з ділянки пам'яті

Так, як і в попередній роботі, будемо виводити числа по 15 в одному рядку. Для переміщення по ділянці пам'яті використовується вказівник та адресна арифметика, лістинг 7.9.

Лістинг 7.9 – Функція виведення чисел на консоль

```
//Функція роздруковує числа з ділянки пам'яті по 15 в рядку
void printData(int *data, int size) {
    int *end = data + size;
    for(int *p = data, i = 1; p < end; p++, i++) {
        printf("%d%c", *p, i%15==0 ? '\n' : ' ');
    }
}
```

7.4.1.3 Реалізація частини функції main() для першого завдання

Реалізацію наведено в лістингу 7.10.

Лістинг 7.10 – Реалізація main() для першого завдання

```
puts("\n\t Завдання 1");
puts("Заповнити ділянку пам'ті випадковими числами\n"
     " від 100 до 999 та надрукувати числа з цієї ділянки");
//Виділяємо пам'ять під ділянку пам1чті
int size = 100 + 22, *data = new int[size];
//Наповнюємо ділянку числами
srand(time(NULL) | clock());
fillWithRnd(data, size);
//Друкуємо числа, якими заповнено ділянку
puts("Згенеровані числа:");
printData(data, size);
```

7.4.2 Реалізація завдання 2

Завдання полягає в реалізації універсальної функції для обробки ділянки пам’яті з випадковими числами, до якої передається вказівник на функцію тестування чисел у відповідності варіантом. Функція має сформувати нову ділянку пам’яті з числами, що задовольняють умові і надати доступ до цієї ділянки.

7.4.2.1 Формулювання завдання.

В якості прикладу реалізуймо завдання з таблиці 7.2.

Таблиці 7.2 – Завдання на обробку послідовності цілих чисел

№	Завдання
0	З вибірки випадкових трицифрових цілих вибрати і надрукувати парні числа в яких сусідні цифри мають різну парність.

7.4.2.2 Аналіз завдання

У минулій лабораторній роботі ми вже завдання тестування числа, тобто можемо скористатися функцією, яка тестує число.

Алгоритм функції обробки залишається таким самим. Але реалізація має бути іншою, бо ми вже маємо числа в ділянці пам’яті із відомою адресою.

Також до функції в якості параметра має передаватися вказівник на функцію тестування. А для того, щоб це можна було зробити, треба визначитися з типом функції тестування:

```
typedef int TestNumber(int);
```

Крім того функція має надавати доступ до нової ділянки пам’яті з числами, що задовольняють умові.

З урахуванням вимог, що сформульовані вище, прототипом функції, яка буде обробляти задану ділянку пам’яті, визначимо так:

```
int* selectAndRetNumb(int *data, int size, int &newSize,
                      TestNumber *test);
```

7.4.2.3 Реалізація функція тестування вибірки випадкових чисел

Реалізацію функції наведено в лістингу 7.13.

Лістинг 7.13 – Функція тестування вибірки чисел

```
//Функція, яка приймає вказівник на функцію тестування числа,
//за її допомогою вибирає числа, що проходять тест,
//заповнює цими числами нову ділянку пам'яті
//і повертає вказівник на цю ділянку пам'яті
int *selectAndRetNumb(int *data, int size, int &newSize,
                      TestNumber *test) {
    int *end = data + size;
    newSize = 0;
    //Цикл підрахунку чисел, що проходять тестування
    for(int *p = data ; p < end; p++) {
        if(test(*p)) newSize++;
    }
    //Виділяємо пам'ять для нової ділянки
    int *newData = new int[newSize];
    //Цикл заповнення нової ділянки
    int *pNew = newData;
    for(int *p = data ; p < end; p++) {
        if(test(*p)) {
            *pNew = *p;
            pNew++;
        }
    }
    return newData;
}
```

7.4.2.4 Друга частина функції main()

Реалізацію частини функції main() для третього завдання наведено в лістингу 7.14. Тут використовується вибірка випадкових чисел, яка була сформована в результаті виконання попереднього завдання.

Після виконання завдання пам’ять, яку займала нова ділянка пам’яті, звільняється.

Лістинг 7.14 – Реалізація другої частини функції main()

```
puts("\n\t Завдання 2");
puts("У згенерованій вибірці знайти парні,\n"
     "в яких сусідні цифри мають різну парність");
//Вибираємо числа і визначаємо їх кількість
int newSize, *newData;
newData = selectAndRetNumb(data, size, newSize, testDigits);
puts("Вибрані числа:");
printData(newData, newSize);
//Звільняємо пам'ять
delete[] newData;
```

7.4.3 Реалізація завдання 3

На цьому етапі потрібно 10 разів наповнити ділянку пам’яті випадковими числами і для кожної вибірки чисел обчислити і надрукувати числову характеристику у відповідності з варіантом.

7.4.3.1 Формулювання завдання.

В якості прикладу реалізуймо завдання з таблиці 7.1.

Таблиці 7.1 – Завдання на обробку послідовності цілих чисел

№	Завдання
0	У послідовності випадкових трицифрових цілих чисел знайти найбільшу різницю між наступним і попереднім числом.

7.4.3.2 Аналіз завдання

У минулій лабораторній роботі ми вже вирішували аналогічне завдання, тобто алгоритм обчислення характеристики залишається таким самим. Але реалізація має бути іншою, бо ми вже маємо числа в ділянці пам’яті із відомою адресою. Прототип функції, яка буде обробляти задану ділянку пам’яті і повертати обчислений результат, буде виглядати так:

```
int maxDif(int *data, int size);
```

Після реалізації цієї функції можна в функції main() у циклі for наповнювати вже отриману раніше ділянку пам’яті випадковими числами і друкувати результат обробки цієї ділянки.

7.4.3.3 Функція обчислення характеристики для вибірки випадкових чисел

Реалізацію функції наведено в лістингу 7.11.

Лістинг 7.15 – Функція обчислення характеристики вибірки чисел

```
//Функція, що знаходить найбільшу різницю між наступним
//і попереднім числом у заданій послідовності
int maxDif(int *data, int size) {
    //Фіксуємо перше число вякості попереднього
    int pred = *data;
    int max = 0; //найбільша різниця між числами
    //Початок області пам'яті за межами ділянки чисел
    int *end = data + size;

    //Цикл обробки починаємо з другого числа
    for(int *p = data +1; p < end; p++) {
        //Обчислюємо різницю
        int dif = *p - pred;
        if(dif > max) max = dif;
        pred = *p;
    }
    return max;
}
```

7.4.3.4 Третя частина функції main()

Реалізацію частини функції main() для другого завдання наведено в лістингу 7.12.

Лістинг 7.16 – Реалізація третьої частини функції main()

```
puts("\n\t Завдання 2");
puts("10 разів обчислити найбільшу різницю між наступним\n"
     "і попереднім числом у вибірці випадкових чисел");
for(int i = 0; i < 10; i++) {
    fillWithRnd(data, size);
    printf("Найбільша різниця %d\n", maxDif(data, size));
}
```

7.4.1 Реалізація завдання 4

Завдання полягає в тім, щоб створити універсальну функцію для виведення на консоль таблиці значень функції, вказівник на яку передається в якості параметра.

Функцію розрахунку за формулою та функцію тестування вхідних даних вже було створено в проєкті до минулої роботи.

Тип функції для розрахунків за формулою можна визначити таким чином:

```
typedef double Task4Func(double);
```

Прототип функції формування таблиці тоді може виглядати так:

```
void printTab(double min, double max, double step, Task4Func*f);
```

Частина функції main() для реалізації цього завдання може виглядати так само, як і в попередній роботі.

Студентам залишається реалізувати функцію printTab.

Зауважимо, що в якості прикладу реалізації, можна використовувати функцію обробки чисел із завдання 3 цієї роботи, лістинг 7.11, а також функцію, яка розглядалась в теоретичній частині, лістинг 7.2.

```
    puts("\n\t Завдання 4");
    puts("Надрукувати таблицю значень заданої функції\n"
         "використовуючи вказівник на функцію");
    double min, max, step;
    //Введення та контроль вхідних даних
    inptTabParm(min, max, step);
    //Заголовок таблиці
    puts("Таблиця значень функції y = 2(e^(-0.1x))sin(0.5x)");
    //Виведення таблиці
    printTab(min, max, step, funcForTab);
```

7.5 Вимоги до звіту

Звіт має бути оформлений з урахуванням вимог ДСТУ та відповідно до зразків звітів з попередніх лабораторних робіт.

7.6 Контрольні питання

– Визначення поняття вказівник.
– Оголошення та ініціалізація вказівників.
– Звернення до даних через вказівники.
– Використання кваліфікатора const для вказівників.
– Адресна арифметика.
– Нетипізовані вказівники.
– Вказівники на функції та їх використання.
– Виділення та звільнення динамічної пам’яті.
– Написати програму обробки ділянки пам’яті за допомогою вказівників.

8 ЛАБОРАТОРНА РОБОТА 8.
РОБОТА З ОДНОВИМІРНИМИ МАСИВАМИ

8.1 Мета роботи

– Ознайомитися з поняттями масив.
– Навчитися оголошувати та ініціалізувати одновимірні масиви.
– Познайомитися з типовими задачами обробки масивів.
– Навчитися оперувати масивами як параметрами функцій.
– Створити проект для обробки масивів.

8.2 Завдання на роботу

В лабораторній роботі слід створити проект, в якому, відповідно до варіанту з таблиці 8.1, реалізувати такі завдання:

– створення масиву заданим в таблиці 8.1способом з **використанням індексів** і виведення створеного масиву на консоль за допомогою спеціально створеної для цього функції. Масив має складатися з цілих чисел від 10 до 99. Розмір масиву має дорівнювати дню народження, але бути не менше 10 і менше 20. Для нормування розміру до дати народження додавати або віднімати 10**;**

– визначити числову характеристики для створеного масиву, відповідно до варіанту з таблиці 8.1, за допомогою функції, яка обчислює цю характеристику;

– формування нового масиву на основі створеного відповідно до варіанту з таблиці 8.1. Новий масив створити у динамічній пам’яті, за допомогою **функції, яка має повертати вказівник на цей масив.** Отриманий масив надрукувати за допомогою функції, яку було створено на етапі реалізації першого завдання.

Таблиця 8.1 – Завдання на роботу з масивами

№	Створення масиву	Числові характеристики	Формування нового масиву
1	2	3	4
1	Random	Розмах елементів (max-min)	Сформувати масив з парних елементів старого масиву
2	По елементам	Різниця між сумами елементів у парних та непарних позиціях	Сформувати масив з елементів в зворотному порядку
3	Random	Кількість елементів, що перевищують середнє геометричне	Сформувати масив в якому всі елементи перевищують середнє геометричне

Продовження таблиці 8.1

1	2	3	4
4	По елементам	Різниця між середніми арифметичним та геометричним	Видалення заданого елементу із масиву
5	Random	Різниця між сумами парних та непарних елементів	Задане число циклічних зсувів ліворуч
6	По елементам	Середнє арифметичне модуля відхилень елементів масиву від середнього арифметичного	Вставка суми елементів у початок масиву
7	Random	Різниця між середніми арифметичними значеннями парних та непарних елементів	Задане число циклічних зсувів праворуч
8	По елементам	Середнє арифметичне значення модуля різниці між сусідніми елементами масиву	Вставка середнього арифметичного значення в середину масиву
9	Random	Кількість елементів, що перевищують середнє арифметичне значення	Переформатовати масив, спочатку непарні, потім парні, не порушуючи їх порядок слідування.
10	По елементам	Сума елементів, що перевищують середнє арифметичне значення	Додати мінімальне значення у початок масиву, а максимальне у кінець

8.3 Теоретична частина

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.

8.3.1 Оголошення масиву та звернення до його елементів

Оголошення одновимірного масиву має вигляд, представлений на рисунку 8.1.

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

Рисунок 18.1 – 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.

Фактично, оголошення масиву – це виділення ділянки пам’яті для певної кількості даних заданого типу. Але ця ділянка пам’яті виділяється не в «купі», а в області пам’яті, що пов’язана з функцією, де оголошується масив.

Ім’я масиву в цьому випадку є вказівником-константою, тому для нього не можна використовувати операції присвоєння, або збільшувати чи зменшувати його значення. Виділена пам’ять автоматично звільнюється після завершення роботи функції.

Що стосується обробки цієї ділянки пам’яті, то тут можна використовувати усі операції, які ми застосовували для обробки ділянок пам’яті, що були отримані через new, у попередній лабораторній роботі.

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

```
int  ar[10] ;
```

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

```
int  ar [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.

Якщо в 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  ar[] ={2, 5, 10, 3, 6, 0, 9, 4, 5, 7};
```

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 ar[i]. Індекс і в даному прикладі є цілим числом у діапазоні від 0 до 9. Таким чином, ar [0] – це ім'я першого елемента, і т. д, ar [9] – ім'я останнього елемента.

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. Можна звернутися до «елемента масиву» з будь яким номером, але отримати невідомо що. Ще гірше може бути, якщо змінити значення такого «елемента масиву». Наслідки можуть бути катастрофічними для програми.

8.3.2 Приклад використання одновимірного масиву

Нижче наведено приклад створення масиву, який містить числа Фібоначчі. Два перших числа Фібоначчі дорівнюють 0 та 1, а кожне наступне є сумою двох попередніх. У програмі перші 2 елементи масиву заповнюються до циклу, решта – в циклі.

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

8.3.3 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 як тип елементів масиву.

Слід також пам'ятати, що ім'я масиву є вказівником на перший елементу масиву.

Крім того, слід брати до уваги той факт, що масив «не знає», скільки у нього елементів, тому до функції слід передавати і кількість елементів масиву, що має бути оброблено. 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 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

Щоправда, функція може повертати вказівник на масив, і таким чином проблема повернення масиву теж вирішується:

```
int *fibo(int size){
    int *ar = new int[size];
    ar[0] = 0; ar[1] = 1;
    for (int i = 2 ; i < size; i++)
        ar[i] = ar[i-1] + ar[i-2];
    return ar;
}
```

При цьому слід пам’ятати, що має сенс повертати вказівник тільки на масиві, для якого пам’ять було виділено через new.

8.3.4 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 індексу елементу масиву за його значенням;
– пошук максимального та мінімального елементів масиву та їх індексів;
– пошук суми елементів масиву;
– видалення елементу з масиву;
– переворот масиву;

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 масив не знає свого розміру, а оголошений розмір масиву зазвичай перевищує кількість даних у ньому.

8.3.4.1 Функція введення масиву з консолі по елементам

Це найпростіший варіант, який полягає у послідовному введенні з консолі заданої кількості елементів масиву за допомогою об'єкту cin, лістинг 8.1.

Лістинг 8.1 – Функція введення масиву з консолі

```
//Функція для наповнення масиву числами, які вводяться з консолі
void fillArFromConsol(int ar[], int size){
    printf("Введіть %d чисел через пробіл\n", size);
    for(int i = 0; i < size; i++)
        cin >> ar[i];
    //Буфер для введення зайвих символів,
    //які користувач може ввести помилково
    char buf[80];
    gets(buf);
}
```

До функції треба передати вказівник на масив, який треба заповнити і кількість елементів. Елементи масиву вводяться у циклі for.

Зважаючи на те, що cin вводить дані до пробілу, можна набрати числа масиву в одному рядку через пробіл і натиснути Enter. Якщо чисел буде менше, ніж потрібно, функція буде чекати на подальше введення. Якщо чисел буде більше, то зайві будуть зчитані в тимчасовий буфер і не вплинуть на подальшу роботу програми.

8.3.4.2 Функція формування випадкового масиву

Ця функція дещо відрізняється від функції генерації вибірки випадкових чисел, що була розглянута у попередній роботі. Різниця полягає у тому, що до елементів масиву звертаємось через індекси. Також до функції передаємо параметри, які визначають діапазон чисел. Текст функції наведено у лістингу 8.2.

Лістинг 8.2 – Функція наповнення масиву випадковими числами

```
//Функція для наповнення масиву випадковими числами
//min та max визначають діапазон чисел
//Перед використанням функції доцільно викликати srand()
void fillRndAr(int ar[], int size, int min, int max){
    for(int i = 0; i < size; i++)
        ar[i] = min + rand()%(max - min + 1);
}
```

8.3.4.3 Функція виведення масиву на консоль

Ця функція досить проста і не потребує коментарів. Текст функції наведено лістингу 8.3.

Лістинг 8.3 – Функція виведення масиву на консоль

```
//Функція виводить на консоль числа масиву по 15 в рядку
void printArOnConsol(int ar[], int size){
    for(int i = 0; i < size; i++)
        printf("%d%c", ar[i], (i+1)%15 ? ' ' :  '\n' );
    puts("");
}
```

8.3.4.4 Функція пошуку індексу елемента масиву за його значенням

Функція послідовно, в циклі for, переглядає елементи масиву і якщо знаходить заданий елемент, повертає його індекс. Якщо не знаходить, повертає -1, лістинг 8.4.

Лістинг 8.4 – Пошук індексу елемента масиву

```
//Функція знаходить індекс заданого елемента масиву
int arIdx(int x, int ar[], int size) {
    for(int i = 0; i < size; i++) {
        if(ar[i] == x) return i;
    }
    return -1;
}
```

8.3.4.5 Пошук максимального і мінімального елементів масиву та їх індексів

Функція буде повертати результати через параметри. Початкові значення параметрів встановлюються відповідно значенню початкового елемента масиву. Код функції наведено в лістингу 8.5.

Лістинг 8.5 – Функція пошуку максимального і мінімального елемента масиву та їх індексів

```
//Функція пошуку максимального і мінімального значень
//серед елементів масиву та їх індексів
void arMinMaxAndIdx(int ar[], int size,
                    int &min, int &max,int &idxMin, int &idxMax)
{
    min = max = ar[0];
    idxMin = idxMax = 0;
    for(int i = 0; i < size; i++) {
        if(ar[i] > max) {
            max = ar[i];
            idxMax = i;
        }
```

```
        else if(ar[i] < min) {
            min = ar[i];
            idxMin = i;
        }
    }
}
```

8.3.4.6 Функція видалення елемента з масиву

У цій функції елементи масиву переглядаються з кінця, 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 початок масиву.

Перегляд починається з кінця тому, що після видалення елемента шляхом зсуву ліворуч змінюються позиції тільки тих елементів, які вже було переглянуті. Текст функції наведемо у лістингу 8.7.

Лістинг 8.7 – Видалення елементів з масиву

```
void delFromArr(int el, int arr[], int& size){
    for (int i = size-1; 0 <= i; i--){
        if (arr[i] == el){
            size--;
            for (int j = i; j < size ; j++){
                arr[j] = arr[j + 1];
            }
        }
    }
}
```

8.3.4.7 Функція перевороту масиву

Текст функції наведено в лістингу 8.8.

Лістинг 8.8 – Функція перевороту масиву

```
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;
    }
}
```

У цій функції елементи масиву, що симетрично розташовані відносно середини, починаючи з першого і останнього, міняються місцями. Тому треба перебирати тільки половину масиву.

8.3.5 Функції формування нового масиву на основі існуючого

Якщо функція має сформувати новий масив, то постає питання, де виділяти пам’ять під новий масив.

Краще виділяти пам’ять у зовнішній функції і передавати ім’я масиву, для якого виділено пам’ять та його розмір, до функції, яка буде наповнювати цей масив. Але такий варіант прийнятний, якщо наперед відомо, якого розміру має бути новий масив.

Якщо розмір нового масиву невідомий, тоді доводиться виділяти пам’ять для нового масиву у функції, яка його формує. І ця функція має повертати і масив і його розмір. Тут може виникнути проблема звільнення пам’яті, бо зовнішня функція може не знати, як виділялась пам’ять – через new, чи за допомогою malloc.

Розглянемо два приклади.

8.3.5.1 Функція формування масиву накопичених значень елементів.

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

Оскільки розмір нового масиву відомий, то до функції краще передавати ім’я нового масиву, для якого вже виділено пам’ять.

Текст функції наведено в лістингу 8.9.

Лістинг 8.9 – Функція формування масиву накопичених значень

```
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];
    }
}
```

8.3.5.2 Функція, що створює новий масив з невідомим наперед розміром

В якості прикладу розглянемо функцію, що створює новий масив з чисел, які кратні числу 3. Кількість таких чисел невідома, тому спочатку підрахуємо скільки в базовому масиві таких чисел. Після цього можна виділяти пам’ять для нового масиву і наповнювати його числами.

Вказівник на новий масив повернемо через return, а розмір через параметр функції.

Реалізацію функції наведено в лістингу 8.10.

Лістинг 8.10 – Функція створення масиву невідомого розміру

```
//Функція створює новий масив з чисел, що кратні 3
//Повертає вказівник на новий масив і його розмір newSize
int *createAr3(int ar[], int size, int &newSize) {
    //Спочатку рахуємо, скільки є елементів, що кратні 3
    int count = 0;
    for(int i = 0; i < size; i++)
        if(ar[i]%3 == 0) count++;
    //Виділяємо пам'ять для нового масиву
    int *newAr = new int[count];
    //Заповнюємо новий масив
    newSize = 0;
    for(int i = 0; i < size; i++){
        if(ar[i]%3 == 0) {
            newAr[newSize] = ar[i];
            newSize++;
        }
    }
    return newAr;
}
```

8.4 Рекомендації до виконання роботи

Створімо проєкт для лабораторної роботи за допомогою шаблону. Доцільно зразу ж додати підключення заголовних файлів <time.h>, <cmath>.

В якості прикладу будемо розглядати варіант з таблиці 8.2

Таблиця 8.2 – Варіант для прикладу виконання роботи

№	Створення масиву	Числові характеристики	Формування нового масиву
0	Random та по елементам	Добуток усіх елементів	Сформувати масив з елементів старого масиву, які кратні трьом

8.4.1 Реалізація завдання 1

На цьому етапі потрібно створити масив і вивести його на консоль. Для реалізації цього завдання треба створити дві функції – функцію для створення масиву з урахуванням варіанту і функцію для виведення масиву на консоль.

8.4.1.1 Аналіз завдання

Масив має складатися з цілих чисел від 10 до 99.

Відповідно до дати народження 22.08.1943 кількість елементів масиву має дорівнювати 22 - 10 = 12 чисел типу int.

Функції, які потрібно створити розглянуті в теоретичній частині, то залишається написати частину функції main().

8.4.1.2 Реалізація частини функції main() для першого завдання

Приклад реалізації цієї частини main() наведено для обох варіантів у лістингу 8.11.

Лістинг 8.11 – Реалізація main() для першого завдання

```
puts("\n\t Завдання 1");
int size = 12;
int ar[size];

//Заповнення масиву з консолі
fillArFromConsol(ar, size);
puts("Числа введені з консолі");
printArOnConsol(ar, size);

//Заповнення масиву випадковими числами
srand(time(NULL) | clock());
fillRndAr(ar, size, 10, 99);
puts("Випадкові числа");
printArOnConsol(ar, size);
```

8.4.2 Реалізація завдання 2

На цьому етапі потрібно написати функцію, яка повертає числову характеристику елементів масиву. Для варіанту, що розглядається в якості прикладу, це обчислення добутку елементів масиву.

8.4.2.1 Аналіз завдання

Добуток цілих чисел теж буде цілим, але може бути дуже великим, і його значення може перевищити розмір будь-якого цілого типу. Тому приймемо для результату тип long double. Тоді прототип функції буде виглядати так:

```
long double dobutok(int ar[], int size);
```

8.4.2.2 Реалізація функції підрахунку добутку чисел масиву

Реалізацію функції наведено в лістингу 8.12.

В реалізації функції додано перевірку елементів масиву на 0. Адже якщо ми не пропустимо такий елемент, то добуток буде також нульовим.

Лістинг 8.12 – Функція обчислення добутку чисел масиву

```
//Функція обчислює добуток елементів масиву
// які не дорівнюють нулю
long double dobutok(int ar[], int size) {
    long double dob = 1;
    for(int i = 0; i < size; i++) {
```

```
        if(ar[i] != 0)
            dob = dob * ar[i];
    }
    return dob;
}
```

8.4.2.3 Реалізація частини функції main() для другого завдання

Реалізацію цієї частини функції main() наведено в лістингу 8.13.

Лістинг 8.13 – Друга частина функції main()

```
puts("\n\t Завдання 2");
puts("Обчислити добуток  елементів масиву більших за 0");
long double dob = dobutok(ar, size);
printf("Добуток = %1.25Lg\n", dob);
```

8.4.3 Реалізація завдання 3

На цьому етапі потрібно написати функцію, яка на основі заданого масиву створює новий.

8.4.3.1 Аналіз завдання

Відповідно до варіанту потрібно створити новий масив з чисел, як кратні трьом.

Таку функцію реалізовано в теоретичній частині, лістинг 8.10. Функцію для роздруківки масиву ми теж вже створили на першому етапі. То ж залишається звернення до цих функцій додати до функції main()

8.4.3.2 Реалізація частини функції main() для третього завдання

Реалізацію цієї частини функції main() наведено в лістингу 8.14.

Лістинг 8.14 – Остання частина функції main()

```
puts("\n\t Завдання 3");
puts("Створити новий масив з елементів кратних 3");
int newSize;
int * newAr = createAr3(ar, size, newSize);
puts("Масив з чисел кратних трьом");
printArOnConsol(newAr, newSize);
delete[] newAr;
```

8.5 Вимоги до звіту

Звіт має бути оформлений з урахуванням вимог ДСТУ та відповідно до зразків звітів з попередніх лабораторних робіт.

8.6 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.
– Написати функцію відповідно до одного із варіантів індивідуальних завдань до лабораторної роботи.

9 ЛАБОРАТОРНА РОБОТА 9. ВПОРЯДКОВАНІ МАСИВИ

9.1 Мета роботи

– Ознайомитися з поняттям впорядкування масиву.
– Ознайомитися з простими алгоритмами сортування масивів.
– Застосувати вказівник на функцію для реалізації алгоритму сортування.
– Ознайомитися з операціями над впорядкованими масивами.
– Створити програму, яка впорядковує та обробляє масив відповідно до індивідуального завдання.

9.2 Завдання на лабораторну роботу

Створити проєкт в якому буде згенеровано масив, відсортовано його за простим та ускладненим правилами, об’єднано з іншим впорядкованим масивом, та для об’єднаного масиву виконано операції пошуку індексу елемента, видалення та вставки елемента. Метод сортування та ускладнене правило вибирати з таблиці 9.1. Просте правило сортування – це на збільшення для непарних варіантів, а для парних варіантів на зменшення.

Таблиця 9.1 – Варіанти завдань на лабораторну роботу

№	Метод сортування	Ускладнене правило сортування
1	Вибором	Спочатку парні, потім непарні, усі на зростання
2	Обміном	Спочатку парні, потім непарні, усі на спадання
3	Вставкою	Спочатку непарні, потім парні, усі на зростання
4	Вибором	Спочатку непарні, потім парні, усі на спадання
5	Обміном	Спочатку непарні на зростання, потім парні на спадання
6	Вставкою	Спочатку непарні на спадання, потім парні числа на зростання
7	Обміном	Спочатку парні на зростання, потім непарні на спадання
8	Вибором	Спочатку парні на спадання, потім непарні числа на зростання
9	Вставкою	Спочатку від’ємні, потім додатні усі на збільшення
10	Обміном	Спочатку від’ємні на зменшення, потім додатні на збільшення

9.2.1 Вимоги до масивів

Масиви мають складатися з цілих чисел від -50 до 50. Розмір першого масиву має дорівнювати дню народження, але бути не менше 10 і менше 20. Для нормування розміру до дати народження додавати або віднімати 10**.** Розмір другого масиву вдвічі менший за розмір першого.

9.2.2 Вимоги до переліку функцій

У програмі мають бути реалізовані такі функції:

– функція генерації масиву з випадкових чисел заданого діапазону;

– функція виведення масиву на консоль через пробіл;

– функція порівняння елементів масиву для сортування на зростання або на спадання (залежно від варіанту);

– функція порівняння елементів масиву для сортування за ускладненим правилом;

– функція сортування масиву заданим для варіанта методом з використанням вказівника на функцію порівняння елементів масиву;

– функція злиття двох масивів з використанням вказівника на функцію порівняння елементів масиву;

– функція пошуку індексу заданого елемента з використанням вказівника на функцію порівняння елементів масиву

– функція видалення елемента із впорядкованого масиву з використанням вказівника на функцію порівняння;

– функція вставки у впорядкований масив з використанням пошуку позиції для вставки і вказівника на функцію порівняння.

9.2.3 Вимоги до функції main()

У функції main() мають послідовно вирішуватися такі завдання з виведення результату:

– виділення пам’яті для першого масиву;

– наповнення першого масиву числами;

– сортування на зростання або спадання в залежності від варіанту;

– сортування за ускладненим правилом;

– виділення пам’яті для другого та об’єднаного масивів з урахуванням розширення;

– наповнення другого масиву числами;

– сортування за ускладненим правилом;

– злиття першого масиву з другим;

– вставка додаткового елемента у розширений масив із збереженням порядку;

– виведення масиву з додатковим елементом.

– пошук позиції вставленого числа у впорядкованому масиві і виведення інформації про позицію на консоль (але не з функції пошуку позиції);

– видалення цього числа із впорядкованого масиву за простим правилом;

Після кожної операції з масивами виводити їх на консоль з відповідним заголовком.

9.3 Теоретична частина

Сортування, або впорядкування, це процес перестановки елементів масиву з метою отримання певного порядку. Порядок визначається умовою, яка обчислюється для двох елементів масиву. Якщо умова виконується – елементи розташовані правильно. Якщо ні, то треба міняти розташування елементів.

Задача впорядкування (і не тільки масивів) постає дуже часто. Зокрема, впорядкування суттєво прискорює пошук.

На сьогодні існує багато різноманітних алгоритмів сортування. В лабораторній роботі ми розглянемо декілька найпростіших із них.

9.3.1 Використання вказівника на функцію порівняння елементів у функціях сортування масивsd

Результат сортування залежить від правила порівняння елементів масиву. Але правило порівняння не впливає на алгоритм сортування. Тому доцільно у функції сортування передавати в якості параметра вказівник на функцію порівняння елементів масиву.

Така функція приймає два значення повертає і повертає результат порівняння у вигляді цілого числа. Якщо елементи однакові, функція повертає 0. Якщо значення параметрів в порядку їх розташування в списку параметрів відповідають правилу сортування, функція повертає позитивне число. Якщо ж порядок розташування параметрів не відповідає правилу – функція повертає від’ємне число.

Для прикладу розглянемо дві такі функції.

Перша, лістинг 9.1, реалізує правило, відповідно до якого попередній елемент не більше наступного. Якщо застосувати таке правило, то значення елементів масиву після сортування будуть у зростаючому порядку.

Лістинг 9.1 – Функція порівняння елементів, що забезпечує сортування у зростаючому порядку

```
//функція тестування на зроствння
int testGrow(int x1, int x2) {
    return x2 - x1;
}
```

Друга функція, лістинг 9.2, забезпечує впорядкування елементів за

складним правилом – спочатку елементи, які кратні трьом, на зростання, потім решта на спадання.

Лістинг 9.2 – Функція порівняння елементів за складним правилом

```
//функція тестування за складним правилом
//спочатку кратні 3 на зростання, потім решта на спадання
//якщо повертає мінус, порядок порушено
int testHard(int x1, int x2) {
    if(x1%3==0 && x2%3 !=0 || x2%3==0 && x1%3 !=0)
        return abs(x2%3) - abs(x1%3);
    else if(x1%3 == 0)
        return x2 - x1;
    else
        return x1 - x2;
}
```

Легко побачити, що функції мають схожі сигнатури, тож для них можна визначити спільний тип, назвемо його Test:

```
typedef int Test(int x1, int x2);
```

Цей тип ми будемо використовувати для передачі посилань на функції порівняння елементів до функцій сортування.

9.3.2 Алгоритми сортування масивів

9.3.2.1 Методо вибору

Алгоритм сортування елементів масиву за методом вибору можна описати так:

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

Аналіз описаних вище дій показує, що для програмної реалізації цього методу сортування буде потрібно два цикли.

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

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

Якщо в результаті порівняння знаходиться елемент, що менший ніж зафіксований, то порівнювані елементи міняються місцями.

Функцію, яка реалізує описаний алгоритм, наведено в лістингу 9.3.

Лістинг 9.3 – Функція сортування елементів масиву методом вибору

```
//Сортування вибором1
void sortChoise1(int ar[], int size, Test *test) {
    for(int i = 0; i < size-1; i++) {
        for(int j = i+1; j < size; j++) {
            if(test(ar[i], ar[j]) < 0) {
                int t = ar[i];
                ar[i] = ar[j];
                ar[j] = t;
            }
        }
    }
}
```

Наведена функція не єдиний варіант реалізації методу вибору. Для зменшення кількості операцій перезапису можна у внутрішньому циклі тільки запам’ятовувати індекс кращого елемента. А обмін робити після завершення внутрішнього циклу. Саме такий спосіб реалізовано у лістингу 9.4.

Лістинг 9.4 – Інша реалізація методу вибору

```
//Сортування вибором2
void sortChoice2(int ar[], int size, Test *test) {
    for(int i = 0; i < size-1; i++) {
        int bestIdx = i;
        for(int j = i+1; j < size; j++) {
            if(test(ar[bestIdx], ar[j]) < 0)
                bestIdx = j ;
        }
        if(bestIdx != i) {
            int t = ar[i];
            ar[i] = ar[bestIdx];
            ar[bestIdx] = t;
        }
    }
}
```

9.3.2.2 Сортування масиву методом обміну

Алгоритм заснований на принципі порівняння сусідніх елементів та обміну значеннями, якщо їх розташування не відповідає правилу порівняння. Процес починається з першого елемента і закінчується передостаннім. Після першого проходу, найгірший елемент виявиться на останньому місці, бо де б він не був спочатку, після кожного порівняння він буде зміщуватися праворуч. А кращі елементі змістяться на одну позицію ліворуч.

У наступному проході все повторюється заново, але останній елемент

перевіряти вже не потрібно, тому процес завершується на перед передостанньому елементі. А найгірший серед переглянутих елементів виявляється на передостанньому місці.

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

Цей метод отримав ще назву «метод бульбашки». Назва стає зрозумілою, якщо розташовувати масив вертикально У процесі реалізації алгоритму елементи з кращими значенням просуваються до початку масиву (спливають), а елементи з гіршими значенням переміщуються у кінець масиву (тонуть).

Реалізацію цього методу наведено в лістингу 9.5.

Лістинг 9.5 – Функція сортування методом бульбашки

```
//Сортування бульбашкою
void sortBubl(int ar[], int size, Test *test) {
    for(int k = size - 1; k > 0; k--) {
        for(int i = 0; i < k; i++) {
            if(test(ar[i], ar[i+1]) < 0) {
                int t = ar[i];
                ar[i] = ar[i+1];
                ar[i+1] = t;
            }
        }
    }
}
```

Розглянута функція робить проходи по масиву незалежно від результатів сортування. Навіть, якщо масив вже впорядкований, функція буде робити задану кількість проходів.

Але інколи початкове розташування елементів може бути таким, що масив можна відсортувати за меншу кількість проходів. Ознакою впорядкованості масиву на деякій ділянці є відсутність обмінів. Тому для скорочення кількості проходів по масиву, можна зафіксувати індекс останнього елемента, який приймав участь в обміні. В наступному проході має сенс розглядати тільки ті елементи, які мають менший індекс

У функції, яку наведено в лістингу 9.6, для фіксації цього індексу використовується змінна lastXchange, яка використовується для зміни значень лічильника циклів k.

Перед кожним переглядом масиву її значення дорівнює 0. Але, якщо елементи міняються місцями, то їй присвоюється значення поточної позиції в масиві.

Лістинг 9.6 – Другий варіант сортування бульбашкою

```
//Сортування бульбашкою 2
void sortXchange(int ar[], int size, Test *test) {
    int k = size - 1, lastXchange;
    while(k > 0) {
        lastXchange = 0;
        for(int i = 0; i < k; i++) {
            if(test(ar[i], ar[i+1]) < 0) {
                int t = ar[i];
                ar[i] = ar[i+1];
                ar[i+1] = t;
                lastXchange = i;
            }
        }
        k = lastXchange;
    }
}
```

9.3.2.3 Сортування масиву за методом вставки

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

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

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

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

Функція сортування, як свідчить лістинг 9.7 має два цикли.

У зовнішньому циклі послідовно змінюється номер лівої межі невпорядкованою області від другого елементу (i=1) до кінця масиву.

У тілі цього циклу порівнюються елементи, що знаходяться по обидва боки від межі, що розділяє впорядковану і невпорядковану частини.

Якщо порядок порушений, то перший елемент невпорядкованою послідовності запам’ятовується у змінній t, внаслідок чого звільняється місце для зсувів упорядкованих елементів вправо.

Лістинг 9.7 – Функція сортування вставкою

```
//Сортування вставкою
void sortInsert(int ar[], int size, Test *test) {
    for(int i = 1; i < size; i++) {
        if(test(ar[i-1], ar[i]) < 0) {
            int t = ar[i];
            int j =i;
            do {
                ar[j] = ar[j-1];
                j--;
            } while (j>0 && test(ar[j-1],t) <0);
            ar[j] =t;
        }
    }
}
```

Внутрішній цикл do..while забезпечує послідовні зсуви упорядкованих елементів вправо, починаючи з останнього, поки не буде знайдено місце для елемента, що зберігався у змінній t.

Можливість вставки елементу визначається однією з двох умов.

– j==0, тобто елемент t виявився меншим ніж усі елементи впорядкованої частини і має бути поставлений на початку впорядкованої частини масиву.

– (1 < j < i) та значення елементів ar[j-1] та t відповідають умовам сортування.

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

Ефективність роботи цієї функції можна дещо підвищити за рахунок зменшення кількості порівнянь під час пошуку місця для елементу із буфера. Місце вставки для елементу із буфера у впорядкованій частині можна знайти методом дихотомії. Більше того, можна просто скористатися функцією вставки елемента в упорядкований масив, яка буде використовувати дихотомію.

Ці функції буде розглянуто далі, а функцію сортування, яка використовує функцію вставки наведено в лістингу 9.8.

Лістинг 9.8 – Сортування вставкою з використанням функції вставки

```
//Сортування вставкою 2
void sortInsert2(int ar[], int size, Test *test) {
    for(int i = 1; i < size; i++) {
        if(test(ar[i-1], ar[i]) < 0) {
            int leftPartSize = i;
            addToSortArray(ar[i], ar, leftPartSize, test);
        }
    }
}
```

9.3.3 Обробка впорядкованих масивів

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

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

Нижче розглядаються функції, які вирішують ці завдання.

9.3.3.1 Злиття впорядкованих масивів

Ця операція є основою алгоритму сортування злиттям. Але алгоритм сортування злиттям ми поки що розглядати не будемо, а розглянемо просто злиття двох впорядкованих масивів.

Функцію злиття впорядкованих масивів наведено в лістингу 9.9.

Лістинг 9.9 – Злиття двох впорядкованих масивів

```
//Злиття двох впорядкованих масивів
void join(int ar1[], int size1, int ar2[], int size2,
          int ar3[], int &size3, Test*test){
    int i=0,j = 0, k = 0; size3 = size1 + size2;
    while(i < size1 && j <  size2) {
        //Записуємо менший з двох елементів
        if(test(ar1[i], ar2[j]) >= 0)
            ar3[k++] = ar1[i++];
        else ar3[k++] = ar2[j++];
    }
    //Додаємо залишки одного з масивів
    while(i < size1)
        ar3[k++] = ar1[i++];
    while(j < size2)
        ar3[k++] = ar2[j++];
}
```

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 закінчяться 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 циклів while, один з яких не буде виконуватися, тому інший масив вже вичерпався.

Функція також повертає через параметр розмір третього масиву.

9.3.3.2 Пошук позиції елемента у впорядкованому масиві

Для пошуку позиції елемента в упорядкованому масиві використовується 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. Якщо елемента масиву в середині області пошуку більший ніж зразок, то пошук слід продовжувати у лівій половині області пошуку. Якщо більший зразок, тоді шукати треба у правій половині області. Таким чином, в результаті одного порівняння область пошуку звужується наполовину.

Цикл пошуку повторюється доти, поки потрібний елемент не буде знайдений або ширина області пошуку звузиться до нуля, що буде свідчити про те, що потрібного елементу в масиві немає. Якщо елемент не знайдено функція повертає -1,

Код функції наведено в лістингу 9.10.

Лістинг 9.10 – Функція пошуку елемента в упорядкованому масиві

```
//Пошук позиції елемента в упорядкованому масиві
int findPoz(int element, int ar[], int size, Test *test) {
    // left і right - ліва та права межі області рошуку
    int left = 0, right = size -1;
    int poz;
    while(left <= right) {
        //Обчислюємо індекс середини області пошуку
        poz = (left + right)/2;
        if(test(element,ar[poz]) == 0)
            return poz;
        if(test(element,ar[poz]) > 0)
            right = poz -1;//потрібний елемент ліворуч
        else left = poz +1;//потрібний елемент праворуч
    }
    //якщо не знайшли
    return -1;
}
```

9.3.3.3 Вставка елемента у впорядкований масив

Функція вставки елемента до впорядкованого масиву вже використовувалася у другому варіанті сортування вставкою, лістинг 9.8 Тепер розглянемо її реалізацію, лістинг 9.11.

Лістинг 9.11 – Функція вставки елемента в упорядкований масив

```
//Вставка елемента в упорядкований масив
void addToSortAr(int element, int ar[], int &size, Test *test)
    //Перевірку починаємо з кінця масиву
    int i = size-1;
    while(i>=0 && test(element,ar[i])>0){
        //Зсув i-го елемента праворуч
        ar[i+1] = ar[i];
        i--;
    }
    //Вставка
    ar[i+1] = element;
    size++;
}
```

Алгоритм вставки полягає у послідовному аналізі елементів масиву, починаючи з останнього. Якщо цей елемент більший ніж той, що потрібно вставити, його 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. І в масиві має бути достатньо місця для розширення.

Але функцію, наведену в лістингу 9.11 можна дещо прискорити за рахунок зменшення кількості порівнянь, використавши метод дихотомії для пошуку місця вставки.

Функція пошуку позиції для вставки, лістинг 9.12, дуже схожа на функцію пошуку позиції елемента.

Лістинг 9.12 – Функція пошуку позиції для вставки у впорядкований масив

```
//Пошук позиції для вставки методом діхотомії
int pozForInsert(int element, int ar[], int &size, Test *test) {
    // left і right - ліва та права межі області рошуку
    int left = 0, right = size -1;
    int poz;
    while(left <= right) {
        //Обчислюємо індекс середини області пошуку
        poz = (left + right)/2;
        if(test(element,ar[poz]) > 0)
            right = poz -1;//потрібна позиція ліворуч
        else left = poz +1;//потрібна позиція праворуч
    }
    //Індекс позиції вставки буде rigt +1
    return right +1;
}
```

Після знаходження місця вставки можна частину масиву, яка

починається з місця вставки, зсунути праворуч на одну позицію за допомогою функції memmove, що також прискорить роботу функції.

Такий варіант функції вставки наведено в лістингу 9.13.

Як бачимо, спроба підвищити ефективність алгоритму приводить до його ускладнення.

Лістинг 9.13 – Другий варіант функції вставки

```
//Вставка елемента в упорядкований масив
void addToSortArray(int element, int ar[],
                    int &size, Test *test) {
    //Спочатку пошук позиції вставки методом діхотомії
    int poz = pozForInsert(element, ar, size, test);
    //Зсув елементів масиву праворуч
    memmove(ar+poz+1, ar+poz,(size-poz)*sizeof(int));
    //Вставка
    ar[poz] = element;
    size++;
}
```

9.3.3.4 Видалення елемента з упорядкованого масиву

Операція видалення елемента з масиву вже розглядалась в лабораторній роботі 8, лістинг 8.7, але вона не враховує той факт, що масив впорядкований.

Видалення з впорядкованого масиву можна зробити ефективніше, якщо використати функцію пошуку позиції елемента, а потім скористатися функцією зсуву ділянки пам’яті.

Але масив може містити двійників елемента, що видаляється. Цю проблему можна вирішувати різними способами.

В лістингу 9.14 наведено код функції, яка спочатку шукає сусідів знайденого елемента, а потім видаляє їх усіх.

Робота функції починається з пошуку індексу елемента, що видаляється.

Після визначення цього індексу аналізуються елементи ліворуч і праворуч від знайденого і таким чином визначаються границі області з елементами, що підлягають вилученню.

Далі, за допомогою функції memmove, права частина масиву зсувається на місце елементів, що видаляються.

Значення змінної, в якій зберігається кількість елементів, зменшується на кількість видалених елементів.

Лістинг 9.13 – Функція видалення елемента з упорядкованого масиву

```
//Видалення елемента з упорядкованого масиву
//Повертає кількість видалених елементів
int delFromSortArr(int element, int ar[], int &size, Test*test){
    int poz, nDel = 0;
    poz = findPoz(element, ar, size, test);
    if(poz < 0) return nDel;
    //Можливо є двійнки ліворуч
    int left = poz;
    while(left > 0 && ar[left-1] == element )left--;
    //Можливо є двійнки праворуч
    int right = poz;
    while(right < size-1 && ar[right+1] == element )right++;
    nDel = right - left +1;
    //Видалення зсувом
    memmove(ar+left, ar+right+1, (size - right -1)*sizeof(int));
    size-=nDel;
    return nDel;
}
```

В лістингу 9.14 наведено функцію, яка в циклі знаходить позицію елемента і видаляє його шляхом зсуву ділянки масиву на одну позиці. Цикл продовжується поки знаходяться елементи для видалення.

Лістинг 9.14 – Другий варіант функції видалення

```
//Видалення елемента з упорядкованого масиву
//Повертає кількість видалених елементів
int delFromSortAr2(int element, int ar[], int &size, Test*test){
    int poz, cnt = 0;
    while((poz = findPoz(element, ar, size, test))!=-1){
        size--;
        memmove(ar+poz, ar+poz+1, (size - poz)*sizeof(int));
        cnt++;
    }
    return cnt;
}
```

9.4 Рекомендації до виконання роботи

Створити проєкт за допомогою шаблону.

Підключити заголовний файл <time.h>, який потрібен для ініціалізації генератора випадкових чисел.

Визначити тип для функцій порівняння елементів масивів:

```
    typedef int Test(int x1, int x2);
```

Додати до проєкту функції створення випадкового масиву і функцію виведення масиву на консоль, які розглядалися у попередній роботі. Написати першу частину функції main() і протестувати функції.

Створити просту функцію порівняння елементів і функцію сортування

відповідно до варіанту. Реалізація заданого у варіанті методу сортування може бути будь-якою, але обов'язково з використанням вказівника на функцію порівняння. Написати наступну частину функції main і протестувати.

Створити функцію порівняння за складним правилом і протестувати її, написавши наступну частину функції main().

Реалізувати функцію злиття впорядкованих масивів і протестувати її в наступній частині функції main(), створивши і впорядкувавши за складним правилом другий масив і виділивши додаткову пам'ять для третього масиву. Пам'яті треба виділяти на 1 більше, ніж кількість елементів, яка в ньому буде після злиття. Це потрібно для того, щоб було місце для вставки.

Реалізувати функцію вставки елемента в упорядкований масив і протестувати її, написавши чергову частину main().

Реалізувати функцію пошуку і протестувати її.

Реалізувати функцію видалення і протестувати її.

Результати виконання лабораторної роботи мають виглядати приблизно так, як показано в лістингу 9.14.

Лістинг 9.14 – Зразок представлення результатів роботи

```
          Розробник Бивойно П.Г., гр.КІ-231
          Дата народження 1943/08/22.
          Номер у списку групи 13.
          Лабораторна робота № 9.
          Завдання:
Створити проєкт в якому буде згенеровано масив,
відсортовано його за простим та складним правилами,
об'єднано з іншим впорядкованим масивом та для
об'єднаного масиву виконано операції вставки,
пошуку індекса та видалення
Перший масив до сортування
-9 -42 -1 19 34 -26 -36 -44 50 -32 -39 42
Сортування на зростання
-43 -34 -30 -26 -11 -10 -5 -1 1 3 11 28
Спочату кратні 3 на зростання, решта на спадання
-42 -39 -36 -9 42 50 34 19 -1 -26 -32 -44
Другий масив до сортування
34 -16 46 3 -25 1
Спочату кратні 3 на зростання, решта на спадання
3 46 34 1 -16 -25
Після злиття
-42 -39 -36 -9 3 42 50 46 34 34 19 1 -1 -16 -25
-26 -32 -44
Введіть елемент для вставки
34
Масив після вставки
-42 -39 -36 -9 3 42 50 46 34 34 34 19 1 -1 -16
-25 -26 -32 -44
Елемент 34 знаходиться в 9 позиції
Масив після видалення вставленого
-42 -39 -36 -9 3 42 50 46 19 1 -1 -16 -25 -26 -32
```

```
-44
Видалено 3 елементів
```

9.5 Вимоги до звіту

Звіт має бути оформлений з урахуванням вимог ДСТУ та відповідно до зразків звітів з попередніх лабораторних робіт.

9.6 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.

10 ЛАБОРАТОРНА РОБОТА 10. РЯДКИ СИМВОЛІВ

10.1 Мета роботи

– Ознайомитися з поняттям рядок символів.

– Ознайомитися з деякими алгоритмами обробки рядків символів.

– Ознайомитися з деякими стандартними функціями обробки рядків символів.

– Створити програму, яка працює з рядками символів відповідно до індивідуального завдання.

10.2 Завдання на лабораторну роботу

В лабораторній роботі слід створити проект, де мають бути написані і протестовані в методі main() такі функції:

– Функція, що тестує символ і виводить на консоль інформацію про нього, використовуючи функції таблиці, зображеної на рисунку 10.3.

– Функція, що повертає найбільше число із рядка символів, у якому знаходяться числа розділені пробілами. Для вирішення задачі використовувати функції strtok() та atoi() і приклад, наведений на сторінці 154 підручника.

– Функція, яка змінює рядок символів відповідно до вимог варіанту з таблиці 10.1, без використання стандартних функцій для обробки рядків символів.

Таблиця 10.1 – Завдання на обробку рядків символів

№	Завдання
1	Залишити тільки задану кількість символів від кінця рядка
2	Вилучити всі символи до заданої позиції.
3	Вилучити у рядку цифрові символи
4	Залишити у рядку тільки цифрові символи
5	Вилучити початкові пробіли з рядка символів
6	Вилучити хвостові пробіли з рядка символів
7	Перетворити усі великі літери в маленькі
8	Перетворити усі маленькі літери у великі
9	Поміняти маленькі літери на великі, а великі на маленькі
10	Вилучити задану кількість символів, починаючи від заданої позиції

10.3 Теоретична частина

У мові С не було спеціального типу для оголошення рядків символів, у С++ 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 символів.

Масив символів – це послідовність байтів в оперативній пам'яті, де у кожному байті зберігається символ. Кожний байт (символ) має свій порядковий номер, який зветься індексом. Нумерація починається з 0.

Для масиву символів в оперативній пам'яті виділяється ділянка, розмір якої на один байт більший, ніж кількість символів у рядку. Цей додатковий байт використовується для збереження ознаки кінця рядка. У якості такої ознаки використовується символ '\0'.

10.3.1 Оголошення рядка

Символьні рядки, які є змінними програми, оголошуються, як масиви типу char, рисунок 10.1.

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

Рисунок 10.1 – Синтаксис оголошення рядка символів

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

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

Можливо також оголошення через вказівник на ділянку пам'яті типу char:

```
char  *str = new char[100] ;
```

Незалежно від способу оголошення ім'я рядка символів є вказівником типу char на ділянку пам'яті, яку займає рядок.

Одночасно із оголошенням рядок символів можна ініціалізувати. Незалежно від того, скільки символів було ініціалізовано при оголошенні, пам'ять виділяється стільки, скільки задано в оголошенні. Значення символів, що не були ініціалізовані – невизначені («сміття»). Але якщо масив визначений як глобальний чи статичний, то невизначені символи дорівнюють нулю.

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 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  str1[] = "Hello!";
```

Можна також об'єднати з ініціалізацією оголошення рядка через вказівник:

```
char  *str2 = "Вітаю!";
```

Можна оголошувати рядки і в динамічній пам'яті:

```
char  *str2 = new char[80];
```

Але з ініціалізацією тут трохи складніше:

```
    char  *str3 = strdup("Вітаю!");
    free(str3);
```

Тут функція strdup за допомогою функції malloc створює в динамічні пам'яті копію рядкової константи і повертає вказівник на створену ділянку.

Можна ініціалізувати і так:

```
    char  *str4 = new char[7];
    memcpy(str4,"привіт",7);
    delete[]str4;
```

10.3.2 Особливості зчитування рядків символів

Рядок символів можна ввести з консолі за допомогою об'єкту cin так само, як і число. Але якщо рядок містить пробіл, то об'єкту cin буде вважати, що рядок закінчився. Тому для зчитування рядків з консолі краще використовувати метод getLine об'єкту cin, наприклад:

```
char str[80];
cin.getline(str,80);
```

Першим параметром цього методу є посилання на рядок символів, а другим – максимальна кількість символів, що буде прийнята з консолі.

Для того, щоб не виникало проблем з введенням, треба для рядra виділяти достатньо пам'яті, наприклад, як у наведеному вище прикладі.

Рядок можна зчитати і за допомогою функцій gets(), або sanf(), але ці функції не контролюють кількість введених символів, що може призвести до непередбачуваних результатів у разі введення більшої кількості символів, ніж виділено пам'яті.

10.3.3 Кодування символів у рядках

Для кодування символів у мові С використовується застарілий міжнародний стандарт ASCII (American Standard Code for Information Interchange).

За цим стандартом:

– кожен символ займає 1 байт;

– всього є 256 символів;

– усі символи знаходяться у спеціальній таблиці;

– кожен символ має свій внутрішній код, який співпадає з порядковим номером символу в таблиці;

– перша половина цієї таблиці (символи з номерами з 0 по 127) стандартна. Вона містить цифри, латинські літери та інші важливі символи;

– у другій половині таблиці (символи з номерами з 128 по 255) знаходяться літери національних алфавітів та додаткові символи. Ця частина таблиці різна для різних мов та різних кодових таблиць, встановлених у операційній системі.

На рисунку 10.2 наведений фрагмент цієї таблиці, скопійований із підручника [3], стр.381, де її можна побачити її цілком.

ASCII-коди графічних символів (коди 32.. 127)

Символ	10-й код	8-й код	16-й код	Символ	10-й код	8-й код	16-й код
пробіл	32	40	20	F	70	106	46
!	33	41	21	G	71	107	47
"	34	42	22	H	72	110	48
#	35	43	23	I	73	111	49
$	36	44	24	J	74	112	4A
%	37	45	25	K	75	113	4B
&	38	46	26	L	76	114	4C
'	39	47	27	M	77	115	4D
(	40	50	28	N	78	116	4E
)	41	51	29	O	79	117	4F
*	42	52	2A	P	80	120	50
+	43	53	2B	Q	81	121	51
,	44	54	2C	R	82	122	52
-	45	55	2D	S	83	123	53
.	46	56	2E	T	84	124	54
/	47	57	2F	U	85	125	55
0	48	60	30	V	86	126	56
1	49	61	31	W	87	127	57
2	50	62	32	X	88	130	58
3	51	63	33	Y	89	131	59

Рисунок 10.2 – Таблиця кодів символів

10.3.4 Реалізація функцій для обробка рядків символів

Нижче перелічені деякі з найбільш поширених завдань з обробки рядків:

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

Кожне з наведених завдань передбачає роботу з окремими символами рядка. Отримати доступ до символу рядка можна за допомогою операції [] – квадратні дужки. Ця операція застосовується до імені рядка, а в дужках записується номер символу, до якого ми хочемо отримати доступ.

Фактично тут неявно використовується адресна арифметика. Ім'я рядка є вказівником на його початок, а індекс є відстанню від початку до потрібного рядка. Якщо оголошено рядок char *str = "Hello!", то вираз str[1] поверне символ 'e', а операція присвоєння str[1] = 'a' перетворить рядок у "Hallo!".

Ті самі дії можна реалізувати через вказівники. Вираз *(str+1) поверне символ 'a', а операція присвоєння *(str + 1) = 'e' поверне рядку початковий вигляд "Hello!".

Далі, в якості прикладів, наведено декілька функцій обробки рядків.

10.3.4.1 Функція, що обчислює кількість символів у рядку

У цій функції, лістинг 10.1, використовується та особливість рядка, що він закінчується символом '\0'.

Лістинг 10.1 – Підрахунок кількості символів рядка

```
int length(char *str){
    int count = 0;
    while(str[count] != '\0')
        count ++;
    return count;
}
```

10.3.4.2 Функція копіювання частини рядка

До функції, лістинг 10.2, передається рядок, з якого будуть копіюватися символи, та адреса ділянки пам'яті, куди будуть копіюватися потрібні символи. Передається також позиція `index`, з якої треба починати копіювання, та число `count` , яке визначає, скільки символів потрібно скопіювати .

У наведеній функції спочатку перевіряється коректність переданих параметрів і, в разі потреби, вони корегуються. Після цього реалізується послідовне копіювання символів.

Лістинг 10.2 – Копіювання частини рядка

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

10.3.4.3 Функція знаходження позиції групи символів у рядку

До функції, лістинг 10.3, передається рядок `str` , у якому буде йти пошук позиції, та група символів `sub`, позиція якої нас цікавить.

Для визначення довжини рядків використовуємо функцію length, яка була наведена вище у цьому підрозділі.

Лістинг 10.3 – Пошук позиції групи символів у рядку

```
int posInStr(char str[], char sub[]){
  for(int i=0; i<=length(str)-length(sub); i++){
    int j=0;
    while(str[i+j]==sub[j] && j<length(sub))
      j++;
    if (j==length(sub))
      return i;
  }
  return -1;
}
```

У функції послідовно порівнюються ланцюжки символів від кожного поточного символу рядка `str` із відповідними символами рядка `sub`. Якщо усі символи групи збіглися, повертаємо позицію поточного символу рядка `str`, якщо ні, переходимо до аналізу ланцюжка від наступного символу.

10.3.4.4 Функція, що вилучає із рядка зайві пробіли

Функція залишає лише один пробіл там, де їх декілька поспіль і повертає кількість вилучених пробілів, лістинг 10.4.

Вилучення пробілів реалізується шляхом зсуву символів правої частини рядка ліворуч на місце зайвих пробілів і перенесення символу кінця рядка.

Лістинг 10.4 – Функція вилучення зайвих пробілів

```
//Функція замінює багато пробілів на один
//і повертає кількість вилучених пробілів
int delExtraBlank(char *str){
    int count = 0;
    for(int i = 0; str[i] != '\0'; i++){
        if(str[i] == ' ' && str[i+1] == ' ' ){
            i++; //Позиція після першого пробілу
```

```
            int p = i; //Шукаємо позицію не пробілу (p)
            while(str[p] == ' '){
                count++; p++;
            }
            //Переносимо символи ліворуч, поки не скопіюємо \0
            for(int j=0; str[p+j-1] != '\0'; j++)
                str[i+j] = str[p+j];
        }
    }
    return count;
}
```

10.3.4.5 Перетворення цілого числа в рядок символів

Комп'ютер оперує з числами, як даними числових типів. Але для користувачів результати обчислень мають бути представлені у вигляді рядків символів. Так само і зворотний обмін числами. Користувач вводить числа як рядок символів, а цей рядок потрібно привести до числового типу.

Проблема таких перетворень полегшується завдяки тому, що числовий код десяткової цифри менше від коду відповідного символу на 48 (код символу для цифри 0).

В лістингу 10.5 наведено функцію перетворення цілого числа у рядок символів.

Лістинг 10.5 – Перетворення цілого числа в рядок символів

```
char* intToStr(int x) {
    //Рахуємо кількість цифр
    int cntDigit = 1;
    for(int num = x; (num/=10) != 0; cntDigit++);
    //Виділяємо пам1ять для рядка
    int size = (x < 0) ? cntDigit+1 : cntDigit;
    char *str = new char[size+1];
    str[size] = '\0';
    //Заносимо символи цифр з кінця рядка
    for(int i=1, num = abs(x); i <= cntDigit; i++) {
        str[size-i] = num%10 + '0';//виділяємо останню цифру
        num/=10;//вилучаємо останню цифру
    }
    if(x<0) str[0] = '-';
    return str;
}
```

Спочатку підраховується кількість цифр, після чого виділяється пам'ять для рядка з урахування знака для від'ємних чисел.

В кінець рядка заносимо ознаку кінця рядка.

Далі, в циклі, за допомогою операції %10, виокремлюються цифри числа, починаючи з останньої, і перетворюються у код відповідного символу шляхом додавання числа 48 (символе '0'). Отриманий символ і заноситься в рядок також з кінця.

Якщо число було від'ємне, то до початку рядка додається символ мінус.

10.3.4.6 Перетворення рядка символів у ціле число

Ця функція, лістинг 10.6, обробляє символи числа до появи першого нецифрового символу.

Код символу перетворюється у код числа шляхом зменшення на 48.

Лістинг 10.6 –Перетворення рядка символів у ціле число

```
int strToInt(char s[]){
    int x=0, i=0;
    if(s[0] == '-' || s[0] == '+') i=1;
    for( ; (s[i] >= '0'&& s[i] <= '9'); i++ )
        x = x * 10 + (s[i] - 48);
    return s[0]=='-' ? -x : x;
}
```

Значення числа накопичується у змінній x за схемою Горнера, яка використовується для швидкого обчислення значень поліномів.

Наприклад число 24378 може бути представлено таким чином:

24378 = ((((0*10+2)*10+4)*10+3)*10+7)*10)+8

Використання цієї схеми дозволяє накопичувати значення числа у циклі.

10.3.5 Стандартні функції для роботи з рядків символів

Стандартна бібліотека мови С містить багато функцій, які швидко вирішувати різноманітні задачі з обробки рядків символів. Ці функції можна розділити на три групи.

До першої групи можна віднести функції, що оперують з окремими символами.

Друга група функцій призначена для роботи з рядками символів. Прототипи цих функцій оголошені у заголовному файлі <string.h>.

Третя група функцій призначена для прямих та зворотних перетворень між рядками символів та числами.

10.3.5.1 Стандартні функції, що оперують з окремими символами.

Прототипи цих функцій знаходяться у заголовному файлі <ctype.h>.

Перелік найбільш популярних з цих функцій наведено у таблиці 9.1 підручника[3], сторінка 151. Там можна знайти більше інформації про ці функції і познайомитися з прикладами їх використання.

На рисунку 10.3 наведено фрагмент цієї таблиці 9.1 підручника. Ваше завдання протестувати наведені функції.

Функція	Призначення
	Перевіряє, чи є символ sym :
int isdigit(sym)	десятковою цифрою
int isxdigit(sym)	шістнадцятковою цифрою
int isspace(sym)	пробільним символом (символом пробілу, нового рядка, горизонтальної чи вертикальної табуляції)
int islower(sym)	малою латинською літерою
int isupper(sym)	великою латинською літерою

Рисунок 10.3 – Функції тестування символів

10.3.5.2 Стандартні функції для прямих та зворотних перетворень між рядками символів та числами.

Прототипи цих функцій оголошені у заголовному файлі <stdlib.h>.

Перелік найбільш популярних з них наведено у таблиці 9.3 підручника[3], сторінка 155. Там можна знайти інформацію про ці функції і познайомитися з прикладами їх використання. Деякі з них наведені на рисунку 10.4.

Функції перетворення символьних рядків у числа

Функція	Призначення
int atoi (st);	Виділяє у рядку st перше ціле десяткове число і перетворює його у дане з типом int. Числу може передувати довільна кількість символів пробілу. Кінцем рядка вважається перший символ, що не належить до цифр. Повертає знайдене числове значення у разі успішного перетворення, а в разі помилки – результат не визначений.
long atol (st);	Аналог atoi(), але перетворює рядок у число з типом long.
double atof (st);	Аналог atoi(), але перетворює рядок st у дане з типом double. Число у рядку може бути записане як ціле чи як дійсне у формі з фіксованою або з плаваючою крапкою.

Рисунок 10.4 – Функції перетворення рядків у числа

10.3.5.3 Стандартні функції для роботи з рядками символів.

Прототипи цих функцій оголошені у заголовному файлі <string.h>.

Перелік найбільш популярних з них наведено у таблиці 9.2 підручника[1], сторінка 153. Там можна знайти інформацію про ці функції і познайомитися з прикладами їх використання. За допомогою цих функцій рядками можна маніпулювати, як простими змінними. Зокрема, їх можна порівнювати, об’єднувати, копіювати.

На рисунку 10.5 наведені деякі з них.

Основні функції опрацювання символьних рядків

Функція	Призначення
`char * strcpy (sr, s);`	Копіює рядок `s` (з `'\0'` включно) за адресою, заданою параметром `sr`. Повертає значення `sr` – адресу скопійованого рядка.
`char * strcat (sr, s);`	Долучає рядок `s` (з `'\0'` включно) у кінець рядка `sr`. Повертає значення `sr` – адресу доповненого рядка.
`int strcmp (s1, s2);`	Послідовно порівнює символи рядків `s1` і `s2` як дані з типом `unsigned char`. Повертає ціле число, значення якого: `< 0`, якщо `s1 < s2`; `0`, якщо `s1 == s2`; `> 0`, якщо `s1 > s2`.
`unsigned strlen (s);`	Повертає довжину рядка `s` у символах (`'\0'` не враховується).
`char * strchr (s, sym);`	Перевіряє, чи символ `sym` входить у рядок `s`. Повертає вказівник на перше входження `sym` у `s` або `NULL`, якщо `sym` не зустрічається в рядку `s`.
`char * strstr (s1, s2);`	Перевіряє, чи рядок `s2` входить як підрядок у `s1`. Повертає вказівник на перший символ рядка `s2` у `s1` або `NULL`, якщо рядок `s2` не зустрічається у рядку `s1`.
`char * strtok (sr, s);`	Виділяє в рядку `sr` лексеми, обмежені символами з рядка `s` (детальніший опис наведено далі). Повертає вказівник на виділену лексему або `NULL`.
`char * strdup (s);`	Копіює рядок `s` (з `'\0'` включно) в динамічну пам'ять, попередньо виділивши там ділянку потрібної довжини. Повертає адресу рядка в динамічній пам'яті.

Рисунок 10.5 – Функції для роботи з символьними рядками

10.4 Рекомендації до виконання роботи

10.4.1 Завдання 1

Завдання полягає в реалізації функції для тестування символів і виведення на консоль інформації про кожен символ, використовуючи стандартні функції.

Для тестування цієї функції у функції main() ввести тестовий рядок і в циклі протестувати всі його символи.

У тестовому рядку першим символом має бути перша буква прізвища студента, далі остання цифра місяця народження. Окрім того в рідку має бути (якщо до того ще не було) великий і маленький символ шістнадцяткової цифри, буква кирилиці, пробіл та якийсь інший символ.

10.4.1.1 Аналіз завдання

Функція, що тестує символ, може мати такий прототип:

```
void testSimbol(char ch);
```

В тілі функції можна послідовно застосувати до символу функції з рисунка 10.3 і виводити відповідну інформацію. Якщо символ відноситься до декількох категорій, будемо виводити кожен варіант.

Варіанти будемо нумерувати. Це допоможез'ясувати, чи є символ невідомим.

10.4.1.2 Приклад виведення результатів тестування символів

В лістингу 10.7 наведено приклад виведення результатів тестування.

Лістинг 10.7 – Приклад виведення результатів тестування символів

```
          Завдання 1
Протестувати символи заданого рядка
і вивести на консоль інформацію про кожен символ,
використовуючи стандартні функції
Введіть рядок символів
Б8Df =
Результати тестування рядка:
Символ 'Б'
        1. Невідомий символ
Символ '8'
        1. Десяткова цифра
        2. Шістнадцяткова цифра
Символ 'D'
        1. Латинська літера
        2. Шістнадцяткова цифра
        3. Велика латинська літера
Символ 'f'
        1. Латинська літера
        2. Шістнадцяткова цифра
        3. Мала латинська літера
Символ ' '
        1. Пробільний символ
Символ '='
        1. Невідомий символ
```

10.4.2 Завдання 2

В цьому завданні потрібно в рядку символів з числами, що розділені пробілами, знайти найбільше число

10.4.2.1 Аналіз завдання

У підручнику [3] є приклад, в якому за допомогою функції strtok із рядка символів виокремлюються лексеми. Для вирішення завдання можна взяти за

основу цей приклад, додавши перетворення лексеми в число за допомогою стандартної функції atoi() та пошук максимального числа в послідовності.

10.4.2.2 Реалізація другої частини функції main()

Реалізацію наведено в лістингу 10.8.

Лістинг 10.8 – Реалізація частини функції main() для завдання 3

```
puts("\n\t Завдання 2");
puts("Реалізувати функцію, яка повертає найбільше число\n"
     "із рядка символів, у якому знаходяться числа,\n"
     "що розділені пробілами");
puts("Введіть рядок, де цілі числа розділені пробілами");
cin.getline(str,80);
int maxNum = findMaxInNumberLine(str);
printf("Найбільше число в рядку %d\n",maxNum);
```

10.4.3 Завдання 3

Завдання полягає в реалізації функції для модифікації рядка символів відповідно з варіантом без використання стандартних функцій. Рядок після обробки функцією має змінитися.

Я приклад, розглянемо функцію, яка вилучає з рядка символів усі латинські літери.

10.4.3.1 Аналіз завдання

Визначити, чи є символ латинською літерою можна за допомогою стандартної функції isalpha(), але за умовами завдання її використовувати не можна. Натомість можна порівнювати символ з крайніми символами латинської абетки і таким чином визначати приналежність символу до цієї абетки.

Для вирішення завдання будемо послідовно переглядати символи рядка. Якщо в поточній позиції виявлено латинську літеру, то робимо зсув частини рядка, що знаходиться за поточною позицією, на один символ ліворуч і продовжуємо аналізувати символи з тієї самої позиції.

Якщо символ не латинська літера, переходимо до аналізу наступного символу.

10.4.3.2 Реалізація функції

Код функції наведено в лістингу 10.9.

Лістинг 10.9 – Приклад реалізації функції для третього завдання

```
//Функція, яка вилучає з рядка усі латинські літери
void eraseLatin(char*s) {
    int i=0;
    //Цикл аналізу символів рядка
```

```
    while(s[i] != '\0') {
        // Аналіз на латинську літеру
        if(s[i] >= 'A' && s[i] <= 'Z'
                || s[i] >= 'a' && s[i] <='z') {
            //Зсув правої частини ліворуч на один символ
            for(int j=i; s[j] !='\0'; j++)
                s[j] = s[j+1];
        }
        //Переходимо до наступної позиції
        else i++;
    }
}
```

10.4.3.3 Реалізація третьої частини функції main()

Реалізацію наведено в лістингу 10.10.

Лістинг 10.10 – Реалізація частини функції main() для завдання 3

```
    puts("\n\t Завдання 3");
    puts("Реалізувати функцію, яка вилучає із заданого\n"
         "рядка символів усі латинські літери");
    puts("Введіть рядок, де латинські літери та інші символи");
    cin.getline(str,80);
    eraseLatin(str);
    puts("Результат роботи функції");
    puts(str);
```

10.5 Вимоги до звіту

Звіт має бути оформлений з урахуванням вимог ДСТУ та відповідно до зразків звітів з попередніх лабораторних робіт.

10.6 Контрольні питання

– Оголошення рядка символів і його ініціалізація.
– Зчитування рядків символів.
– Рядки символів як параметри функцій.
– Написати функцію для обробки рядка символів за вказівкою викладача.
– Написати функцію до одного із варіантів індивідуальних завдань.

11 ЛАБОРАТОРНА РОБОТА 11. СТРУКТУРИ

11.1 Мета роботи

– Ознайомитися із поняттям структура.
– Ознайомитися із способами оголошення та ініціалізації структур.
– Опанувати алгоритми обробки структур.
– Створити проект для обробки структур.

11.2 Завдання на лабораторну роботу

В лабораторній роботі слід створити проект для роботи з масивом структур. Поля структури сформувати відповідно до варіанту курсового проекту, перший (не кореневий) рівень розгалуженого дерева. Полів має бути не менше трьох і вони повинні бути різних типів.

У проекті слід реалізувати такі функції:
– перегляд масиву;
– додавання нового елементу;
– вилучення елемента;
– сортування за декількома параметрами.;
– декілька варіантів вибору інформації із масиву.

11.3 Теоретична частина

Структура - це тип даних, якому відповідає суміш елементів 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.

11.3.1 Оголошення шаблону та ініціалізація структур

Синтаксис оголошення шаблону структури виглядає так, як показано на рисунку 11.1.

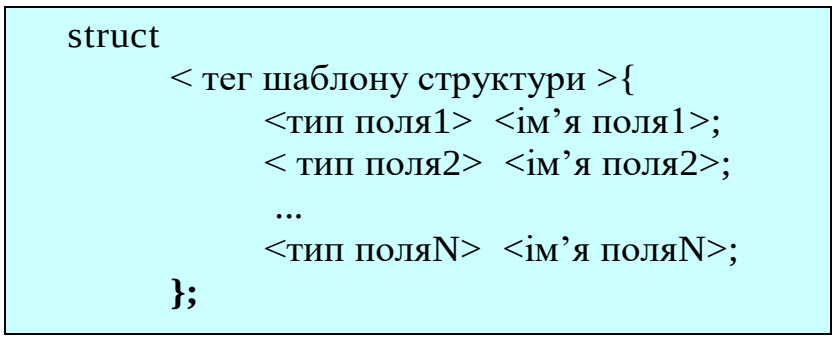


Рисунок 11.1 – Синтаксис оголошення шаблону структури

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-211", 5.00, 0};
```

Для структур можна використовувати оператор присвоювання. У прикладі, що наведено нижче, всім полям запису s1 будуть присвоєні значення полів запису bestStud.

```
      s1 = bestStud;
```

Значення полів числового типу також можна визначати за допомогою оператора присвоєння:

```
s2.srBall = 3.5;
s2.nzd =1;
```

Але для полів типу рядок символів, для яких виділяється ділянка пам’яті визначеного розміру, присвоєння значення допускається тільки через функцію копіювання рядків символів.

```
strncpy(s1.fio, "Мазепа", 20);
```

Полями структури можуть бути також і масиви і структури.

11.3.2 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_MEMO 15
#define GR_MEMO 7
#define AR_SIZE 30
struct Stud {
    char fio[FIO_MEMO];   char gr[GR_MEMO];
    float srBall; int nzd;
};
Stud ar[AR_SIZE]={
   {”Чуб П.П.”,”КС-051”, 4.55, 0},
    {”Гай А.Л.”,”КС-052”, 1.55, 3},
    {”Кит А.В.”,”КС-051”, 2.45, 1},
    {”Кіт С.В.”,”КС-052”, 1.25, 3},
    {”Бут К.Л.”,”КС-052”, 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.

```
strncpy(ar[size].fio, "Петренко А.П.", FIO_MEMO);
strncpy(ar[size].gr, "КС-041", GR_MEMO);
ar[size].srBall = 3.2;
ar[size].nzd = 1;
size++;
```

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';
```

11.3.3 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 масивів, але різниця тут у тому, що елементи можуть мати різні типи, і доступ до елементів проводиться не за індексом, а за назвою.

11.3.4 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 за прізвищем студента. У такій ситуації доцільно для кожного варіанту сортування писати функції порівняння, які будуть передаватися у функцію сортування як параметр.

11.4 Рекомендації до виконання роботи

В якості зразка розглянемо етапи створення проекту «Результати атестації».

У цьому проекті ми будемо працювати з масивом, що містить результати атестації студентів. Кожен елемент масиву буде зберігати структуру, 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:

– сортування за прізвищем студента;

– комплексне сортування за кількістю незадовільних оцінок ↓ + та середньому балу ↑.

– вибірка студентів, які мають середній бал не нижче заданого.

– підрахунок загального числа студентів, що мають незадовільні оцінки.

Даний проект може слугувати прикладом того, як виконати завдання для лабораторної роботи.

11.4.1 Створення допоміжних функцій

Розташуємо ці функції після функції main();

Функція getInt() дозволить уникнути проблем введення даних, що пов'язані з використанням cin. Це функція введення цілого числа, яка також дозволяє контролювати діапазон чисел і виводити текст запиту не введення даних:

```
//Функція введення цілого числа в заданому діапазоні
int getInt(const char* prompt, int min, int max){
  int num;
  string s;
  do{
    printf("%s(%d..%d) ", prompt, min, max);
    fflush(stdin);
    getline(cin,s);
    num = atoi(s.c_str());
  }while(num < min || num > max);
  return num;
}
```

Функція getStr() дозволить уникнути проблем, що пов'язані з використанням cin.getline. Вона дозволяє контролювати довжину рядка, а також виводити текст запиту не введення даних:

```
//Функція для введення рядка символів
char *getStr(const char* prompt, int maxMem) {
    printf("%s ", prompt);
    string s;
    fflush(stdin);
    getline(cin,s);
        if(s.length() >= maxMem)
        s = s.substr(0,maxMem-1);
    int size = s.length() + 1;
    char* cStr = new char[size];
```

```
    strncpy(cStr, s.c_str(), size);
    return cStr;
}
```

Функція confirm() формує запит на підтвердження виконання операції або відмову від неї:

```
// Функція запиту на підтвердження виконання операції
int confirm(const char* question){
  printf("\n%s", question);
  int c = getInt("\nЯкщо так, ведіть 1, інакще 0 ",0 ,1);
  return c == 1;
}
```

Не забуваймо прописати прототипи цих функцій на початку програми.

11.4.2 Визначення глобальних змінних та констант

Оголошення глобальних змінних дозволить використовувати їх у всіх функціях файлу, що спростить структуру функцій проекту, а також формування меню для проєкту.

Ці оголошення робимо перед прототипами функцій.

Оголосимо спочатку константи для розмірів прізвища студента та назви групи:

```
#define FIO_MEMO 15
#define GR_MEMO   7
```

Далі оголосимо шаблон структури:

```
//Шаблон структури
struct Stud {
  char fio[FIO_MEMO];
  char gr[GR_MEMO];
  int srBall;
  int nezd;
};
```

Після цього оголосимо також глобальні змінні для масиву структур, його місткості та кількості записів у масиві:

```
//Глобальні змінні
Stud *ar; //  Вказівник на масив структур у динамічній пам’яті
int capacuty; //Місткість масиву (обсяг виділеної пам'яті)
int size; // Кількість елементів у масиві
```

11.4.3 Початкова ініціалізація та демонстрація масиву

Головну функцію проекту main() ми почнемо з виведення на консоль інформації про тему проєкту та розробника

```
int main(){
```

```
    puts("Проєкт 'Результати атестації'");
    puts("Розробник Бивойно П.Г. гр.ІКС-1988");
```

11.4.3.1 Ініціалізація масиву

Для зручності тестування застосунку одразу проініціалізуємо глобальні змінні та створімо початковий масив структур.

```
    capacity = 6;//Стартова допустима кількість елементів = 6
    size = 5;//Стартова кількість записів у масиві = 5
    ar = new Stud[capacity] {
        {"Чуб П.П.", "ПІ251",  95, 0},
        {"Гай А.Л.", "КІ252",  35, 3},
        {"Кит А.В.", "ПІ251",  55, 1},
        {"Кіт С.В.", "КІ252",  25, 3},
        {"Бутковський К.Л.", "КІ252", 85, 0}
    };
    showAll();
```

В останньому рядку попереднього тексту викликається функція showAll(), яка відображає масив на консолі. Ця функція розглядається в наступному підпункті.

11.4.3.2 Функція відображення масиву на консолі

Алгоритм функції полягає в циклічному виведенні полів кожного з елементів масиву. Для форматування виводу використовуються налаштування функції printf(). Зауважимо, що використання printf() можливо при умові включення заголовного файлу <stdio.h>.

Зірочки у рядках формату означають, що на їх місце буде вставлене число із списку параметрів. У нашому випадку це розміри полів для імені студента та назви групи.

Нижче наводиться код цієї функції:

```
void showAll(){
    cout << "\n Результати атестації:\n" ;
    printf(" %-*s %-*s %s %s\n", FIO_MEMO, "Студент",
                GR_MEMO, "Група", "Бал", "Борги");
    for(int i=0;i<size;i++){
        printf(" %-*s %-*s %3d %3d\n", FIO_MEMO, ar[i].fio,
               GR_MEMO, ar[i].gr, ar[i].srBall, ar[i].nezd);
    }
}
```

Тепер можна протестувати перший крок розробки проєкту.

11.4.4 Інтерфейс користувача для проєкту

Формування інтерфейсу користувача також реалізуймо у головній

функції програми, функції main().

11.4.5 Створення елементів меню

Для створення меню ми скористаємося структурою, яка буде містити посилання на текст пункту меню та вказівник на функцію обробки цього пункту меню.

Оголошення типу для функцій реалізації пунктів меню та оголошення структури для пунктів меню слід розташувати перед прототипами функцій після структури для елементів масиву. Це оголошення може виглядати так:

```
//Тип для функції, що викликається через меню
typedef void MenuFunc();

//Структура для пункту меню
struct MenuUnit{
  // Текст пункту меню
  const char* text;
  //Вказівник на функцію, що викликається цим пунктом
  MenuFunc *func;
};
```

Тип MenuFunc показує, що для реалізації пунктів меню ми будемо використовувати функції, які нічого не повертають і не потребують параметрів

У структурі для пунктів меню маємо поле text яке представляє текст пункту меню та поле funk, яке зберігає вказівник на функцію, що реалізує даний пункт меню.

Створімо і одразу проініціалізуємо масив структур для пунктів меню:

```
    MenuUnit menu[] {
        {"Показати результати атестації", showAll},
        {"Додати запис", addRecord},
        {"Сортувати за прізвищем", sortByName},
        {"Сортувати за успішністю", sortByResult},
        {"У кого середній бал не нижче заданого", showForBall},
        {"Кількість, що мають заборгованості", calcProblem},
        {"Завершити роботу", NULL},
    };
    //Активізація меню
    runMenu(menu,"Меню проєкту");
```

Завершується наведений фрагмент функції main() викликом функції runMenu(), яка активізує створене меню. Ми її розглянемо в наступному підпункті. А поки що завершимо роботу масивом меню.

Кожен елемент цього масиву має у своєму складі посилання на функцію(її ім’я). Функцію showAll ми вже реалізували, для решти слід оголосити прототипи.

```
void addRecord();
void sortByName();
void sortByResult();
```

```
void showForBall();
void calculateProblem();
```

Окрім того, потрібно мати реалізацію цих функцій, бо інакше меню не спрацює. Тому треба написати хоча б пусті реалізації цих функцій. До цього достатньо скопіювати прототипи цих функцій і додати пару фігурних дужок замість крапки з комою.

Але буде краще, якщо вони будуть повідомляти, що до них звернулися. Так, наприклад, як для функції addRecord():

```
//Функція додавання нового запису до масиву
void addRecord() {
    puts("\tДодавання нового запису до масиву");
}
```

11.4.5.1 Функція активізації меню.

Ця функція виводить варіанти роботи програми, та забезпечує введення номера вибраного варіанту і виклик відповідної функції. Ці операції реалізовано у безкінечному циклі:

```
//Функція активізації меню
void runMenu(MenuUnit menu[], const char*title) {
    while(true) {
        printf("\n\t    --- %s ---\n", title);
        int cnt = 0;
        //Виводимо меню
        for( ; ; cnt++) {
            printf("\t%2d. %s\n", cnt + 1, menu[cnt].text);
            if(menu[cnt].func == NULL) break;
        }
        // Вводимо номер варіанту
        int variant = getInt("\tEnter variant number, please ",
                             1, cnt+1);
        if(variant == cnt +1) {
            puts("\n\tРоботу з програмою закінчено.");
            break;
        }
        // Виклик вибраної функції
        menu[variant - 1].func();
    }
}
```

Після реалізації цієї функції можна запустити нашу програму і перевірити, чи працює меню.

На цьому реалізацію функції main() завершено. Далі можна зайнятися реалізацією функцій меню.

11.4.6 Додавання нових даних до масиву структур

Додати нового студента у масив структур дуже просто. Для цього

достатньо ввести з консолі значення кожного поля наступного елемента масиву і збільшити лічильник елементів масиву на 1.

Але ця функція, перш за все, має контролювати наповненість масиву. Якщо масив заповнений, то потрібно виділити для нього нову ділянку пам'яті більшого розміру і переписати туди елементи старого масиву. Тому доцільно написати спеціальну функцію, яка буде виконувати таку роботу. Не забудьте оголосити прототип цієї функції:

```
    //Функція збільшення місткості масиву
    void growAr(){
        //Створюємо ділянку пам'яті, яка більша в півтора разу
        capacity = capacity + capacity / 2;
        Stud* newAr = new Stud[capacity];
        //Копіюємо старий масив в нову ділянку пам1яті
        memcpy(newAr, ar, size * sizeof(Stud));
        Stud *oldAr = ar;//Запам'ятали, щоб звільнити пам'ять
        ar = newAr;
        //Звільнюємо пам'ять, яку займав старий масиив
        delete[] oldAr;
    }
```

Окрім того, процедуру спілкування з користувачем і формування нової структури також доцільно реалізувати окремою функцією. У цій функції ми скористаємось допоміжними функціями geInt() та getStr():

```
//Функція для формування структури Stud
Stud getRecord() {
    Stud stud;
    strcpy(stud.fio, getStr("Введіть прізвище: ",FIO_MEMO));
    strcpy(stud.gr, getStr( "Введіть групу: ",GR_MEMO));
    stud.srBall = getInt("Введіть середній бал ",0,100);
    stud.nezd = getInt("Введіть кількість боргів",0,10);
    return stud;
}
```

Тепер можна дописати функцію, що забезпечить додавання нового елемента до масиву. Нагадаємо, що пусту її реалізацію було створено.

Після додавання нового елемента масив виводиться на консоль.

```
    //Функція меню для додавання нового запису до масиву
    void addRecord() {
        puts("\n\tДодавання нового запису до масиву");
        if(size == capacity) growAr();
        ar[size] = getRecord();
        size++;
        showAll();
    }
```

11.4.7 Видалення запису з масиву

Для реалізації видалення перш за все треба дізнатися, який запис користувач хоче видалити. Після цього потрібно видалити вибраний елемент. Тому задачу доцільно розділити на дві частини.

Спочатку написати функцію, яка буде виводити перенумерований список студентів та повертати індекс вибраного запису. За зразок можна взяти виведення меню у функції runMenu.

Після визначення індексу можна скористатися допоміжною функцією confirm(), яку вже реалізовано.

Для видалення достатньо зробити зсув частини елементів масиву ліворуч, так як ми це робили в лабораторних роботах 8 та 9 і зменшити значення змінної, яка зберігає кількість елементів на 1.

На рисунку 11.2 наведено можливий варіант діалогу програми з користувачем в процесі видалення запису.

```
        Перелік студентів для вибору:
         1. Чуб П.П.
         2. Гай А.Л.
         3. Кит А.В.
         4. Кіт С.В.
         5. Бутковський К.Л.
         6. Не вибирати
        Enter variant number, please (1..6) 5

Вибрано Бутковський К.Л.
Ви дійсно хочете його видалити?
Якщо так, ведіть 1, інакше 0 (0..1) 1

 Результати атестації:
 Студент              Група   Бал Борги
 Чуб П.П.             ПІ251    95   0
 Гай А.Л.             КІ252    35   3
 Кит А.В.             ПІ251    55   1
 Кіт С.В.             КІ252    25   3
```

Рисунок 11.2 – Діалог програми з користувачем при видаленні

11.4.8 Функція сортування масиву структур

Наш проект передбачає декілька варіантів сортування масиву структур. Але функцію сортування ми напишемо одну, а для реалізації потрібних варіантів сортування напишемо відповідні функції порівняння структур, які будемо передавати до функції сортування як параметр.

Для реалізації цього завдання треба перш за все оголосити тип для функції порівняння структур. Додаймо таке оголошення до загальної частини програми після оголошення структури для елементів масиву:

```
//Тип для функції порівняння структур
typedef int Comparator(Stud, Stud);
```

Після цього можна писати функцію сортування. Зважаючи на те, що масив та його розмір у нас оголошено глобальними змінними, можна до функції сортування передавати тільки функцію порівняння структур.

Прототип функції сортування може бути таким:

```
void sort(Comparator *test);
```

А реалізацію функції можна взяти із лабораторної роботи, де впорядковувалися масиви. Єдине, що доведеться поміняти, це параметри масиву – його тип, назва та кількість елементів.

11.4.9 Реалізація сортування за ім'ям студента

Для реалізації цього пункту меню слід спочатку написати і оголосити відповідну функцію порівняння елементів структур. В даному випадку це порівняння імен студентів:

```
//Функція порівняння структур за прізвищем студента
int testByName(Stud s1, Stud s2) {
    return strcmp(s2.fio, s1.fio);
}
```

У наведеному коді використовується стандартна функція порівняння рядків символів, яка повертає результат цілочислового типу. Цей результат буде від'ємний, якщо перший параметр менший, додатній - якщо перший параметр більший, та нуль, якщо параметри однакові.

Після цього нескладно реалізувати і саму функцію меню:

```
//Функція  меню для сортування за прізвищем
void sortByName() {
    puts("\n\tСортування за прізвищем");
    sort(testByName);
    showAll();
}
```

11.4.10 Реалізація сортування за результатами навчання

У даному випадку ми маємо впорядкувати масив за значеннями декількох полів структури. Будемо сортувати масив структур за кількістю «боргів», а при рівності цього показника по середньому балу. Окрім того, значення кількості боргів має зростати, а середній бал зменшуватися. Якщо ж кількість боргів і середній бал однакові, то слід структури впорядкувати за прізвищами.

Відповідна функція порівняння буде виглядати так:

```
//Функція порівняння структур за результатами навчання
int testByResult(Stud s1, Stud s2){
    if(s1.nezd != s2.nezd)
        //По кількості боргів на зростання
        return s2.nezd - s1.nezd;
```

```
    if(s1.srBall != s2.srBall)
        //По балу на спадання
        return s1.srBall - s2.srBall;
    //Якщо показники однакові впорядковуємо заа прізвищем
    return strcmp(s2.fio, s1.fio);
```

А функція реалізації відповідного пункту меню буде такою:

```
void sortByResult(){
    sort(testByResult);
    showAll();
}
```

11.4.11 Вибірка студентів по граничному балу

Це завдання полягає в отриманні списку студентів, що мають середній бал не нижче заданого.

Вирішення задачі мало чим відрізняється від задачі виведення масиву на консоль. Різниця полягає у тому, що потрібно організувати діалог з користувачем і отримати значення граничного балу. Для цього можна скористатися функцією getInt(). Окрім того, масив перед обробкою доцільно відсортувати за результатами навчання. А виводити треба тільки ті структури, що відповідають заданій умові:

```
//Вибірка студентв, що мають бал не нижче заданого
void showForBall() {
    puts("\n\tВибірка по балу");
    int ball = getInt(" Введіть граничний бал ", 1, 100);
    sort(testByResult);
    printf("\nСтуденти, що мають бал не нижче %d:\n", ball);
    printf(" %-*s %-*s %s %s\n", FIO_MEMO, "Студент",
           GR_MEMO, "Група", "Бал", "Борги");
    for(int i=0;i<size;i++){
        if(ar[i].srBall >= ball)
        printf(" %-*s %-*s %3d %3d\n", FIO_MEMO, ar[i].fio,
               GR_MEMO, ar[i].gr, ar[i].srBall, ar[i].nezd);
    }
}
```

11.4.12 Інформація про студентів що мають заборгованості

Вирішення цієї задачі потребує підрахунку кількості елементів у масиві, що відповідають умові, що кількість заборгованостей більша нуля.

Напишемо таку функцію:

```
int calcDebt(){
    int debtCount = 0;
    for(int i = 0; i < size; i++)
        if(ar[i].nezd > 0) debtCount++;
    return debtCount;
}
```

Після цього можна перейти до реалізації функції меню. У цій функції окрім виведення кількості боржників ще надрукуємо інформацію про таких студентів, якщо такі є.

```
//Функція вибірки студентів із заборгованостями
void calcProblem() {
    puts("\n\tПідрахунок проблем");
    int count = calcDebt();
    if(count == 0)
        puts("\nСтудентів із заборгованостями нема.");
    else {
        sort(testByResult);
        printf("\n Усього студентів із боргами %d:\n", count);
        printf(" %-*s %-*s %s %s\n", FIO_MEMO, "Студент",
               GR_MEMO, "Група", "Бал", "Борги");
        for(int i=0; i<size; i++)
            if(ar[i].nezd > 0)
                printf(" %-*s %-*s %3d %3d\n",
                       FIO_MEMO, ar[i].fio,
                       GR_MEMO, ar[i].gr,
                       ar[i].srBall, ar[i].nezd);
    }
}
```

11.5 Вимоги до звіту

Звіт має бути оформлений з урахуванням вимог ДСТУ та відповідно до зразків звітів з попередніх лабораторних робіт.

У звіті обов’язково має бути лістинг з копією консолі після запуску програми і тестування усіх функцій.

11.6 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.

12 ЛАБОРАТОРНА РОБОТА 12. ФАЙЛИ

12.1 Мета роботи

– Ознайомитися з операціями над файлами.
– Створити програму, яка створює та обробляє файл.

12.2 Завдання на лабораторну роботу

В лабораторній роботі слід створити програму, у якій реалізується робота з файлом структур, який створюється із масиву структур, що було сформовано у попередній роботі. Тобто завдання майже те саме, що у попередній роботі, але структури вже зберігаються не у масиві, а у файлі. Можна просто додати до попереднього проекту додаткові функції.

У проекті мають бути реалізовані такі функції:

– створити проєкт як продовження попереднього;
– переписати масив у файл;
– додати запис у кінець файлу не використовуючи масив;
– переписати файл у масив;
– показати протокол змін. Цей пункт передбачає наявність текстового файлу, до якого записується інформація про операції з даними через меню.

12.3 Теоретична частина

Файл – це послідовність байтів, певним чином організована, яка зберігається на **зовнішньому** носії інформації, або надходить від нього.

Дуже часто замість статичного поняття «файл» використовують динамічне поняття «потік даних».

12.3.1 Відкриття та закриття файлу

Файл – це послідовність байтів, яка зберігається на деякому зовнішньому носії інформації, або формується ним. Кожен файл має своє, унікальне ім’я. Повне ім’я файлу (наприклад d:\labs\lab14\test1.txt) включає адресу директорії, де знаходиться файл (d:\labs\lab14), назву файлу (test1) та розширення (txt).

Для роботи з файлами необхідно підключити заголовний файл <stdio.h>.

Вираз «відкрити файл» означає, що програма отримує доступ до потоку даних з (до) цього файлу.

Для відкриття файлу використовується функція fopen(), до якої треба передати два параметри у вигляді вказівників на рядки символів. Перший параметр визначає ім’я файлу, другий – режим роботи з файлом. Функція повертає вказівник на структуру типу FILE, що містить дані про відкритий файл та пов’язаний з ним потік даних. Для спрощення будемо називати цей

вказівник файловою змінною.

Ім’я файлу, що передається функції fopen() може бути повним, або скороченим (без адреси директорії). В разі передачі скороченого імені використовується активна на даний момент директорія.

Режим роботи встановлюється такими специфікаторами, таблиця 12.1

Таблиця 12.1 - Специфікатори режимів

Специфікатор			Значення
«r»	«rb»		**read:** Файл відкривається для зчитування. Файл має існувати.
«w»	«wb»		**write:** Файл створюється для запису. Якщо такий файл існував, його дані втрачаються.
«a»	«ab»		**append:** Файл використовується для доповнення. Нова інформація додається у кінець файлу. Якщо такого файлу не було, він створюється.
«r+»	«rb+»	«r+b»	**read/update:** Файл відкривається для оновлення. Дозволено читання і запис. Файл має існувати.
«w+»	«wb+»	«w+b»	**write/update:** Створюється новий файл для запису і зчитування. Якщо такий файл існував, його дані втрачаються.
«a+»	«ab+»	«a+b»	**append/update:** Файл відкривається для запису і зчитування. Операції запису можливі тільки у кінець. Якщо такого файлу не було, він створюється.

Символ b у визначенні специфікатора означає, що файл відкривається як бінарний. Якщо символ b відсутній, або замість нього написано символ t, тоді файл розглядається як текстовий.

Після завершення роботи з файлом його слід закрити. Для цього використовується функція fclose(), до якої в якості параметра передають вказівник на структуру, що пов’язана з файлом. Нижче наведено приклад відкриття і закриття файлу:

```
char *fileName = "d:\\CodeBlock\\test.txt";
//Відкриваємо файл для запису і читання
FILE *testFile = fopen(fileName, "wt+");
. . .
fclose(logFile);
```

Після виконання цього фрагменту коду на диску d в папці CodeBlock має з’явитися пустий файл test.txt.

12.3.2 Обробка текстових файлів

Інформацію з текстових файлів можна зчитувати символами або рядками. Так само можна і записувати інформацію до текстових файлів.

Обмін інформацією через символи забезпечується функціями fgetc() та

fputc(), а для роботи з рядками використовують функції fgets() та fputs().

До усіх перелічених функцій потрібно передавати параметри.

В якості останнього параметру, потрібно передавати файлову змінну, що визначає звідки (або куди) буде надходити інформація.

Фунція fgetc() не потребує інших параметрів і повертає код прочитаного символу приведеного до типу int.

Фунція fputc() в якості першого параметру приймає код символу, що записується до файлу.

Фунція fputs() в якості першого параметру приймає вказівник на рядок (масиив символів), що має бути записано у файл..

До функції fgets() в якості першого параметру передається вказівник на масив символів, куди буде записано прочитаний рядок. Другим параметром є число, що визначає максимально допустиму кількість прочитаних символів.

Слід зауважити, що із застосуванням перелічених функцій пов'язано багато нюансів. Докладніше про їх використання можна почитати у підручнику, або у довідкових джерелах інформації.

Розглянемо приклад:

```
void example_12_3_2() {
    int len = 80; //максимально довжина рядка
    char buf[len];//буфер для символів, що вводимо
    char *fileName = "d:\\CodeBlock\\test.txt";
    //Відкриваємо файл для запису і читання
    FILE *testFile = fopen(fileName, "wt+");
    //Читаємо рядки з консолі(stdin)і пишемо до файлу
    puts("Вводимо рядки символів до 80 символів");
    puts("Для завершення вводимо пустий рядок");

    while(true) {
        fgets(buf, len, stdin);
        if(strcmp(buf, "\n") == 0)
            break;//вихід з циклу, якщо рядок пустий
        fputs(buf,testFile);
    }
    //Повертаємо файл до початку
    rewind(testFile);
    //Виводимо вміст файлу
    while(fgets(buf,512, testFile)!= NULL) {
        printf("%s",buf);
    }
    fclose(testFile);
}
```

Програма спочатку зчитує рядки символів з консолі і записує ці рядки у відкритий файл. Завершується ця частина програми після того, як буде введено пустий рядок.

Після завершення введення даних файл повертається до початку, і рядки послідовно виводяться на консоль.

Зверніть увагу, що консоль теж розглядається як файл. Файлова змінна

для введення з консолі має стандартну назву stdin, а для виведення stdout.

В кінці програми файл закривається. Якщо його не закривати, то можна продовжувати вводити до нього інформацію.

12.3.3 Обмін блоками даних з файлом

Блок даних являє собою певну кількість байтів, що послідовно розташовані у пам'яті або у файлі. Блоком може бути число будь якого типу. структура, масив. Блок даних може розглядатися також як певна кількість однакових елементів, які, фактично, теж є блоками. Це дуже зручно. Наприклад, якщо у файл записують масив структур, тоді блоком буде масив, а елементом – структура.

Для обміну блоками даних з файлом використовуються функції fwritre() та fread(). Ці функції повертають кількість реально прочитаних, або записаних елементів і мають чотири параметри.

Як і у попередньо розглянутих функціях, останнім параметром є файлова змінна, що визначає файл, з яким відбувається обмін.

Першим параметром цих функцій є безтиповий вказівник на ділянку пам'яті (буфер) з якої зчитуються, або куди записуються дані.

Другим параметром є розмір (у байтах) одного елемента блоку.

Третім параметром є кількість елементів у блоці.

12.3.3.1 Збереження масиву у файлі

Як приклад використання цих функцій розглянемо реалізацію функції збереження масиву структур типу Stud у файлі.

Спочатку ми записуємо у файлі кількість записів(елементів масиву) і після цього пишемо весь масив цілком, одним блоком:

```
void storeToFile() {
    puts(" Введіть ім'я файлу");
    char fileName[80];
    gets(fileName);
    //ВІдкриваємо файл для бінарного запису
    FILE *file = fopen(fileName,"wb");
    //Зберігаємо у файлі кількість елементів масиву
    fwrite(&size, sizeof(int), 1, file);
    //записуємо весь масив
    fwrite(ar, sizeof(Stud),size, file);
    printf("Масив збережено у файлі %s\n", fileName);
    fclose(file);
}
```

12.3.3.2 Відновлення масиву з файлу

Для відновлення масиву із файлу, якщо він був збережений так, як у попередньому прикладі, можна використати таку функцію:

```
void restoreFromFile() {
    puts(" Введіть ім'я файлу");
    char fileName[80];
    gets(fileName);
    FILE *file = fopen(fileName,"rb");
    //читаємо кількість елементів у масиві
    int n;
    fread(&n, sizeof(int), 1, file);
    //Виділяємо динамічну пам'ять для масиву
    Stud *newAr = new Stud[n];
    fread(newAr, sizeof(Stud), n, file);
    fclose(file);
    //Оновлюємо вказівник на масив
    Stud *oldAr = ar;
    ar = newAr;
    delete[] oldAr;
    //Оновлюємо параметри масиву
    size = n;
    capacity = n;
    printf("Array restored from file %s", fileName);
}
```

Тут спочатку зчитується розмір масиву у змінну n, Після чого зчитується блок розміром n структур типу Stud і записується в ділянку пам’яті за адресою newAr, для якої перед цим було виділено пам’ять.

12.3.4 Робота із записами файлу

В деяких випадках нема потреби переписувати всю інформацію із файлу в оперативну пам’ять. Працювати з інформацією можна безпосередньо у файлі. Для цього слід відкрити файл для читання і далі організувати цикл послідовного перегляду записів файлу. При цьому слід враховувати, що після зчитування кожного запису файлова позиція автоматично переходить на початок наступного.

Як приклад, розглянемо функцію пошуку студента з найвищим балом у файлі, який зберігає масив структур Stud:

```
Stud example_12_3_4() {
    puts(" Введіть ім'я файлу");
    char fileName[80];
    gets(fileName);
    //Відкриваємо бінарний файл для читання
    FILE *file = fopen(fileName,"rb");
    //читаємо кількість елементів у масиві
    int n;
    fread(&n, sizeof(int), 1, file);
    Stud best, stud; //Краща і поточна структури файлу
    //Спочатку вважаємо що кращий - це перший у файлі
    fread(&best, sizeof(Stud), 1, file);
```

```
    //Організуємо цикл перегляду решти записів файлу
    while(--n){
        fread(&stud, sizeof(Stud), 1, file);
        if(stud.srBall > best.srBall)
            best = stud;
    }
    return best;
```

12.3.5 Робота з файлом в режимі прямого доступу

У прикладах, що були розглянуті у попередніх пунктах, файл використовувався у режимі послідовного доступу. Дані записувалися або зчитувалися послідовно від початку файлу до кінця.

Але інколи виникає потреба зчитувати інформацію з певного місця у файлі, або змінювати вже існуючі дані. Для забезпечення такої можливості введено поняття вказівника поточної позиції файлу і є функції керування положенням цього вказівника.

Можна вважати, що позиція файлу – це номер байту від початку файлу, з якого починається зчитування або запис. Значення цієї позиції повертає функція ftell(), до якої в якості параметра передається файлова змінна. Можна також скористатися функцією fgetpos(), але ця функція записує позицію у змінну, адреса якої передається в якості параметра.

За допомогою функції fsetpos() можна встановити значення позиції відповідно до значення змінної, адреса якої передається в якості параметра.

Встановити позицію на початок файлу можна функцією rewind().

Для керування позицією файлу також використовується функція fseek(). Першим параметром цієї функції є файлова змінна. А от значення бажаної позиції формується відповідно до значень параметрів offset та base.

Параметр offset визначає відстань позиції файлу від бази. Значення може бути як від'ємним так і додатнім.

Параметр base визначає значення бази, яка розглядається як умовний 0. Можливі значення цього параметру наведені у таблиці 14.2

Таблиця 14.2 – Константи базових значень файлового вказівника

Константа	На яку позицію посилається
SEEK_SET	Початок файлу
SEEK_CUR	Поточна позиція
SEEK_END	Кінець файлу

В якості прикладу використання перелічених функцій розглянемо приклад сортування записів файлу.

Виходячи з того, що у файлі ми не оперуємо поняттям «індекс», для сортування скористаймося методом бульбашки, де можна використовувати поняття попередній (s1) і наступний (s2) замість індексів:

```
//Сортування файлу бульбашкою 2
```

```
void sortFile(Comparator *test) {
    puts(" Введіть ім'я файлу для сортування");
    char fileName[80];
    gets(fileName);
    //Відкриваємо файл для читання і модифікації
    FILE *file = fopen(fileName,"rb+");
    //читаємо кількість записів у файлі
    int size;
    fread(&size, sizeof(int), 1, file);
    //Сортування
    Stud s1,s2; //Послідовні записи у файлі
    int last = size, ok = false;
    do {
        last--;
        ok = true;
        //Пропускаємо перший запис з розміром
        fseek(file, sizeof(int), SEEK_SET);
        //Читаємо перший запис про студента
        fread(&s1, sizeof(Stud), 1, file);
        for(int i = 0; i < last; i++) {
            //Переходимо на поточну позицію файлу
            fseek(file, 0, SEEK_CUR);
            //Читаємо наступний запис
            fread(&s2, sizeof(Stud), 1, file);
            if(test(s1, s2) < 0) {
                //Міняємо місцями записи у файлі
                fseek(file, -2*(sizeof(Stud)), SEEK_CUR);
                fwrite(&s2, sizeof(Stud), 1, file);
                fwrite(&s1, sizeof(Stud), 1, file);
                ok = false;
            }
            else s1 = s2;
            }
    } while(!ok);
    fclose(file);
    puts(" Файл відсортовано");
}
```

У програмі аналізуються два послідовних записи файлу, починаючи з початку. Запис, який був другим на якомусь етапі стає першим у наступному порівнянні. Якщо порядок розташування пари записів не відповідає умовам сортування, записи міняються місцями. Для переходу у потрібну позицію використовується функція fseek().

В результаті одного такого перегляду файлу принаймні один запис (останній) займає правильну позицію. Тому кожний наступний перегляд файлу потребує на одне порівняння менше.

Програма завершує роботу, якщо обмінів записів не було. Для фіксації цього факту використовується логічна змінна ok.

Слід зауважити, що у такий спосіб реально файли не сортують, бо програма буде працювати дуже повільно. Для сортування файлів намагаються

максимально використовувати пам’ять і процедури злиття.

Даний приклад є просто ілюстрацією використання функцій прямого доступу до файлу.

12.4 Рекомендації до виконання роботи

Роботу можна продовжувати з проектом із попередньої роботи.
Далі послідовно розглядаються можливі кроки реалізації завдання.

12.4.1 Розширення переліку глобальних змінних

Одним із завдань роботи є реєстрація операцій з проєктом у текстовому файлі протоколу. З протоколом будуть працювати різні функції проєкту, тому вказівник на цей файл оголосимо як глобальний, за межами функцій:

```
FILE *logFile; //Файл протоколу роботи з програмою
```

12.4.2 Створення меню для роботи з файлом

Для роботи з файлом нам потрібно ще п’ять функцій до вже існуючих восьми. Але за правилами UI/UX дизайну в меню не бажано мати більше 5-7 функцій. Тому до існуючого масиву структур додаємо тільки функцію виклику меню роботи з файлом:

```
        {"Робота з файлом", menuFile},
```

Меню для роботи з файлом сформуємо в окремій функції:

```
//Меню для роботи з файлом
void menuFile(){
   MenuUnit menu[] {
         {"Зберегти масив у файлі", storeToFile},
         {"Додати запис до файлу", addToFile},
         {"Показати вміст файлу", showFile},
         {"Відновити масив з файлу", restoreFromFile},
         {"Показати файл протоколу", showLogFile},
         {"Вийти з меню", NULL},
    };
    //Активізація меню
    runMenu(menu,"Меню роботи з файлом");
}
```

Після цього треба, так само як це робили в попередній роботі, треба прописати прототипи нових функцій, створити для них заготовки і протестувати меню.

12.4.3 Створення файлу протоколу

Далі слід створити файл протоколу і додати до нього інформацію про початок роботи з проєктом. Це доцільно зробити у функції main() перед

викликом меню, щоб у протоколі реєструвалися усі дії.

Після виходу із меню файл протоколу слід закрити.

Відповідний фрагмент функції main() після доопрацювання може виглядати так:

```
logFile = fopen("logByvoino", "wt+");
fprintf_s(logFile, "Проєкт 'Атестація' стартував.");
//Активізація меню
runMenu(menu,"Меню проєкту");
fclose(logFile);
```

12.4.4 Реалізація функції showProtocol

Тепер можна реалізувати функцію showProtocol і протестувати її.

Файл протоколу відкривається в функції main() для запису і читання. Тому у будь-який момент його можна перевести до початку за допомогою функції rewind() і роздрукувати. Закривати файл після цього не треба, бо до нього ще може надходити інформація. Закривається він у функції main() після виходу із головного меню

У пункті 12.1.2 методичних вказівок є приклад роботи з текстовим файлом.

Наприкінці цієї функції слід ще додати виведення повідомлення у файл протоколу.

12.4.5 Функції збереження масиву у файлі

Наступним кроком може бути реалізація функції збереження масиву у файлі. Приклад такої функції наведено у підпункті 12.3.3.1.

Наприкінці цієї функції слід ще додати виведення повідомлення у файл протоколу.

12.4.6 Функція виведення на консоль інформації із файлу

Перш за все слід зауважити, що інформацію слід виводити безпосередньо з файлу, без переписування її у масив.

Для реалізації цієї функції слід відкрити файл для бінарного читання Далі прочитати кількість елементів у масиві. Після цього організувати цикл зчитування записів і виведення інформації на консоль.

Можна скористатися, як зразком, функцією пошуку найкращого студента із пункту 12.3.4 методичних вказівок і функцією showAll для виведення масиву на консоль.

Наприкінці цієї функції слід ще додати виведення повідомлення у файл протоколу.

12.4.7 Функція відновлення масиву із файлу

Приклад такої функції наведено у пункті 14.1.3.

Наприкінці цієї функції слід ще додати виведення повідомлення у файл протоколу.

12.4.8 Функція додавання запису до файлу

Робота починається з введення імені файлу і перевірки, чи він існує. Якщо файл не існує, то його слід створити і записати у початок нуль типу int.

Код перевірки існування файлу, та його створення, вразі відсутності може виглядати так:

```
puts("\n\tДодавання нового запису до файлу");
 char fileName[80];
 int size = 0;
 puts(" Введіть ім'я файлу");
 gets(fileName);
 FILE *file = fopen(fileName,"rb+");
 if(file == NULL) {
     //Створення файлу, якщо він не існує
     file = fopen(fileName,"wb");
     fwrite(&size, sizeof(int),1, file);
     freopen(fileName, "rb+", file);
 } else
     fread(&size,sizeof(int),1, file);
```

Після виконання цього коду ми маємо файл, який відкрито для модифікації і кількість записів у файлі в змінній size.

Після цього треба отримати структуру, яку ми будемо записувати до файлу. Її треба оголосити, і отримати від користувача значення усіх полів. Схожі дії виконувалися у функції додавання записів до масиву, де для цього використовувалася функція getRecord().

Далі можна переходити до модифікації файлу.

Особливість цього завдання полягає в тому, що до файлу треба додати запис і ще поміняти значення кількісті записів у файлі, яке зберігається на початку файлу.

Спочатку можна перейти у кінець файлу і додати новий запис.

Далі збільшити значення кількості записів у файлі на 1 і записати нове значення у початок файлу.

Для переміщень по файлу можна використовувати функції seek та rewind.

Наприкінці цієї функції слід ще додати виведення повідомлення у файл протоколу.

12.5 Вимоги до звіту

Звіт має бути оформлений з урахуванням вимог ДСТУ та відповідно до зразків звітів з попередніх лабораторних робіт.

У звіті обов’язково має бути лістинг з копією консолі після запуску програми і тестування функцій роботи з файлом у такій послідовності:

– запуск програми;
– запис вмісту масиву до файлу;
– додавання нового запису напряму до файлу;
– виведення вмісту файлу на консоль;
– запис вмісту файлу до масиву;
– виведення вмісту масиву на консоль.

12.6 Контрольні питання

– Що таке файл та типи файлів.
– Як відкрити файл та які можуть бути режими роботи з файлом.
– Запис та читання текстів з файлу.
– Обмін блоками даних з файлом.
– Функції прямого доступу до записів файлу.
– Написати функцію для обробки файлу.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Code::Blocks. – Режим доступу: https://en.wikipedia.org/wiki/ Code::Blocks
2. ДСТУ 3008:2015. – Режим доступу: https://science.kname.edu.ua/images/dok/derzhstandart_3008_2015.pdf
3. Шпак З.Я. Програмування мовою С / З.Я. Шпак. – Львів : вид-во НУ «Львівська політехніка», 2011. – 436 с.
4. Code::Blocks download. – Режим доступу: https://sourceforge.net/projects/codeblocks/
5. C++ українською. Встановлення IDE CodeBlocks. – Режим доступу: https://www.youtube.com/watch?v=E4GUEgZQhhw
6. Уроки по C++. Урок №21. Базове форматування коду. . – Режим доступу: https://acode.com.ua/urok-21-bazove-formatuvannya-kodu/
7. Numerics limits. – Режим доступу: https://en.cppreference.com/w/c/types/limits
8. Limits on Floating-Point Constants. – Режим доступу: https://learn.microsoft.com/en-us/cpp/cpp/floating-limits?view=msvc-170
9. C Library math.h Functions. – Режим доступу: https://www.geeksforgeeks.org/c-library-math-h-functions/
10. Diagrams.net. – Режим доступу: https://app.diagrams.net/
11. Kernighan B. W., Ritchie D. M. C Programming Language / http://www.amazon.com/C-Programming-Language-2nd-Edition/dp/0131103628Dennis M. Ritchie
12. C Programming Tutorial. – Режим доступу: https://www.guru99.com/c-programming-tutorial.html