**+МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

Чернігівський національний технологічний університет

ОСНОВИ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторного практикуму та самостійної роботи з дисципліни

«Основи програмування»

для студентів спеціальностей

121 – “Інженерія програмного забезпечення”

123 – “Комп’ютерна інженерія”

ЗАТВЕРДЖЕНО

на засіданні кафедри

інформаційних та комп’ютерних систем

протокол № 1 від 29.08.19

**Чернігів ЧНТУ 2019**

Основи програмування. Методичні вказівки до лабораторного практикуму та самостійної роботи з дисципліни «Основи програмування» для студентів спеціальностей 121 – “Інженерія програмного забезпечення” 123 – “Комп’ютерна інженерія”. /Укл.: Бивойно П.Г. Бивойно Т.П. – Чернігів: ЧНТУ, 2019. – 144 с.

Укладач: Бивойно Павло Георгійович, канд. техн. наук, доцент;

Бивойно Тарас Павлович, ст. викладач;

Відповідальний за випуск: В.М. Базилевич, зав. кафедрою інформаційних та комп’ютерних систем, канд. економ. наук, доцент.

Рецензент: Скітер І.С., канд. техн. наук, доцент кафедри інформаційних технологій та програмної інженерії Чернігівського національного технологічного університету

ЗМІСТ

[1 Знайомство з інтегрованим середовищем розробки “QT-creator” 9](#_Toc25867686)

[1.1 КОРОТКІ ВІДОМОСТІ ПРО “QT-creator” 9](#_Toc25867687)

[1.1.1 Головне меню середовища 10](#_Toc25867688)

[1.1.2 Панель режимів робіт 10](#_Toc25867689)

[1.1.3 Панель доступу до результатів опрацювання проекту 10](#_Toc25867690)

[1.1.4 Створення проекту 10](#_Toc25867691)

[1.1.5 Панель редагування програм 11](#_Toc25867692)

[1.1.6 Функція main() 13](#_Toc25867693)

[1.1.7 Приклад програми з введенням числових даних 14](#_Toc25867694)

[1.1.8 Приклад програми з введенням символьних даних 15](#_Toc25867695)

[1.1.9 Дослідження специфічних констант 15](#_Toc25867696)

[1.1.10 Збереження проекту 16](#_Toc25867697)

[1.2 Завдання для самостійної роботи 17](#_Toc25867698)

[1.3 Вимоги до звіту 17](#_Toc25867699)

[1.4 Контрольні питання 17](#_Toc25867700)

[*Рекомендована література* 17](#_Toc25867701)

[2 Типи даних та розрахунки за формулами 18](#_Toc25867702)

[2.1 Короткі теоретичні відомості 18](#_Toc25867703)

[2.1.1 Змінні і константи 18](#_Toc25867704)

[2.1.2 Типи даних 18](#_Toc25867705)

[2.1.3 Арифметичні типи даних 19](#_Toc25867706)

[2.1.4 Описи змінних 21](#_Toc25867707)

[2.1.5 Операція розміру sizeof 22](#_Toc25867708)

[2.1.6 Арифметичні операції 23](#_Toc25867709)

[2.1.7 Операції присвоєння 23](#_Toc25867710)

[2.1.8 Вирази 24](#_Toc25867711)

[2.1.9 Пріоритети операцій у С++ 25](#_Toc25867712)

[2.1.10 Узгодження типів 25](#_Toc25867713)

[2.1.11 Бібліотека математичних функцій cmath 26](#_Toc25867714)

[2.1.12 Форматне виведення даних через функцію printf() 27](#_Toc25867715)

[2.1.13 Форматне виведення даних через cout 30](#_Toc25867716)

[2.2 Завдання на лабораторну роботу 30](#_Toc25867717)

[2.3 Вимоги до звіту 31](#_Toc25867718)

[2.4 Контрольні питання 31](#_Toc25867719)

[*Рекомендована література* 31](#_Toc25867720)

[3 Функції 32](#_Toc25867721)

[3.1 Короткі теоретичні відомості 32](#_Toc25867722)

[3.1.1 Правила написання функцій 32](#_Toc25867723)

[3.1.2 Виклик функції 33](#_Toc25867724)

[3.1.3 Прототип функції 34](#_Toc25867725)

[3.1.4 Прототипи бібліотечних функцій 34](#_Toc25867726)

[3.1.5 Способи передачі параметрів у функції 35](#_Toc25867727)

[3.1.6 Області оголошення та доступу до імен 36](#_Toc25867728)

[3.1.7 Макроси з параметрами 38](#_Toc25867729)

[3.2 Реализація проекту «function» 38](#_Toc25867730)

[3.2.1 Підключення допоміжних файлів 39](#_Toc25867731)

[3.2.2 Створення макросу з параметрами 39](#_Toc25867732)

[3.2.3 Створення прототипів функцій 40](#_Toc25867733)

[3.2.4 Функція main() програмного файлу 40](#_Toc25867734)

[3.2.5 Функція f1(), що повертає значення 41](#_Toc25867735)

[3.2.6 Функція f2() типу void 42](#_Toc25867736)

[3.2.7 Функція з виведенням проміжних результатів 42](#_Toc25867737)

[3.3 Дослідження створеного проекту 42](#_Toc25867738)

[3.3.1 Дослідження передачі параметрів за посиланням 42](#_Toc25867739)

[3.3.2 Анализ передачи параметров по значению 43](#_Toc25867740)

[3.4 Вимоги до звіту 43](#_Toc25867741)

[3.5 Контрольні питання 43](#_Toc25867742)

[*Рекомендована література* 43](#_Toc25867743)

[4 Логічний тип даних і розгалуження у програмах 44](#_Toc25867744)

[4.1 Короткі теоретичні відомостіІ 44](#_Toc25867745)

[4.1.1 Логічний тип даних 44](#_Toc25867746)

[4.1.2 Алгоритми з розгалуженнями 45](#_Toc25867747)

[4.1.3 Програмування розгалужень 47](#_Toc25867748)

[4.1.4 Оператор переходу goto 51](#_Toc25867749)

[4.2 Реализація проекту «if\_switch» 52](#_Toc25867750)

[4.2.1 Початковий інтерфейс проекту 52](#_Toc25867751)

[4.2.2 Структура проекту 53](#_Toc25867752)

[4.2.3 Функція main() програмного файлу 53](#_Toc25867753)

[4.2.4 Завдання із підручника 54](#_Toc25867754)

[4.3 Вимоги до звіту 54](#_Toc25867755)

[4.4 Контрольні питання 54](#_Toc25867756)

[*Рекомендована література* 54](#_Toc25867757)

[5 Використання циклів „while” і „do...while” в Ітераційних алгоритмах 55](#_Toc25867758)

[5.1 Короткі теоретичні відомості 55](#_Toc25867759)

[5.1.1 Циклічні алгоритми 55](#_Toc25867760)

[5.1.2 Оператор while 55](#_Toc25867761)

[5.1.3 Оператор do…while 57](#_Toc25867762)

[5.1.4 Переривання циклу 58](#_Toc25867763)

[5.1.5 Ітераційні алгоритми 59](#_Toc25867764)

[5.1.1 Функції для обчислення кубічного кореня 60](#_Toc25867765)

[5.2 Завдання для самостійної роботи 61](#_Toc25867766)

[5.3 Вимоги до звіту 62](#_Toc25867767)

[5.4 Контрольні питання 62](#_Toc25867768)

[*Рекомендована література* 63](#_Toc25867769)

[6 Використання циклів „while” і „do...while” для обчислення сум нескінченних рядів 64](#_Toc25867770)

[6.1.1 Алгоритми обчислення сум нескінченних рядів 64](#_Toc25867771)

[6.1.2 Функції для обчислення синуса 65](#_Toc25867772)

[6.2 Завдання для самостійної роботи 66](#_Toc25867773)

[6.3 Вимоги до звіту 67](#_Toc25867774)

[6.1 Контрольні питання 67](#_Toc25867775)

[*Рекомендована література* 67](#_Toc25867776)

[7 Обробка данных за допомогою циклу for 68](#_Toc25867777)

[7.1 Короткі теоретичні відомостіІ 68](#_Toc25867778)

[7.1.1 Оператор циклу for 68](#_Toc25867779)

[7.1.2 Особливості використання циклу for 69](#_Toc25867780)

[7.1.3 Табулювання функцій 70](#_Toc25867781)

[7.1.4 Обробка послідовностей цілих чисел 72](#_Toc25867782)

[7.1.5 Випадкові числа 73](#_Toc25867783)

[7.1.6 Обробка послідовності випадкових чисел 74](#_Toc25867784)

[7.2 Завдання на лабораторну роботу 75](#_Toc25867785)

[7.3 Вимоги до звіту 77](#_Toc25867786)

[7.4 Контрольні питання 77](#_Toc25867787)

[*Рекомендована література* 77](#_Toc25867788)

[8 Основи роботи із вказівниками 78](#_Toc25867789)

[8.1 Короткі теоретичні відомості 78](#_Toc25867790)

[8.1.1 Оголошення та ініціалізація покажчиків 78](#_Toc25867791)

[8.1.2 Звернення до даних через покажчики 79](#_Toc25867792)

[8.1.3 Використання кваліфікатора const для покажчиків 80](#_Toc25867793)

[8.1.4 Адресна арифметика 80](#_Toc25867794)

[8.1.5 Нетипізовані вказівники 81](#_Toc25867795)

[8.2 Завдання для САМОСТІЙНОЇ РОБОТИ 82](#_Toc25867796)

[8.3 Вимоги до звіту 83](#_Toc25867797)

[8.4 Контрольні питання 83](#_Toc25867798)

[*Рекомендована література* 84](#_Toc25867799)

[9 Рядки символів 85](#_Toc25867800)

[9.1 Короткі теоретичні відомості 85](#_Toc25867801)

[9.1.1 Оголошення рядка 85](#_Toc25867802)

[9.1.2 Особливості зчитування рядків символів 86](#_Toc25867803)

[9.1.3 Особливості обробки рядків символів 86](#_Toc25867804)

[9.1.4 Приклади обробки рядків символів 86](#_Toc25867805)

[9.1.5 Функція копіювання частини рядка 87](#_Toc25867806)

[9.1.6 Функція знаходження частини рядка у рядку 87](#_Toc25867807)

[9.1.7 Функція перетворення рядка символів у числовий масив 87](#_Toc25867808)

[9.1.8 Кодування символів у рядках 88](#_Toc25867809)

[9.2 Завдання для роботи 90](#_Toc25867810)

[9.3 Вимоги до звіту 90](#_Toc25867811)

[9.4 Контрольні питання 90](#_Toc25867812)

[*Рекомендована література* 91](#_Toc25867813)

[10 Одновимірні масиви 92](#_Toc25867814)

[10.1 Короткі теоретичні відомості про масиви 92](#_Toc25867815)

[10.1.1 Оголошення одновимірного масиву та звернення до його елементів 92](#_Toc25867816)

[10.1.2 Приклад використання одновимірного масиву 94](#_Toc25867817)

[10.1.3 Одновимірні масиви як параметри функцій 94](#_Toc25867818)

[10.2 Функції обробки масивів чисел 95](#_Toc25867819)

[10.2.1 Функція формування випадкового масиву 95](#_Toc25867820)

[10.2.2 Функція виведення масиву на консоль 96](#_Toc25867821)

[10.2.3 Функція введення масиву з консолі по елементам 96](#_Toc25867822)

[10.2.1 Функція перевороту масиву 97](#_Toc25867823)

[10.2.2 Функція вилучення елементу з масиву 98](#_Toc25867824)

[10.2.3 Функція формування масиву накопичених значень 98](#_Toc25867825)

[10.3 Завдання для САМОСТІЙНОЇ РОБОТИ 99](#_Toc25867826)

[10.4 Вимоги до звіту 100](#_Toc25867827)

[10.5 Контрольні питання 100](#_Toc25867828)

[*Рекомендована література* 100](#_Toc25867829)

[11 Сортування масивів 101](#_Toc25867830)

[11.1 Методи сортування масивів 101](#_Toc25867831)

[11.1.1 Сортування вибором 101](#_Toc25867832)

[11.1.2 Сортування обміном (метод бульбашки) 105](#_Toc25867833)

[11.1.3 Сортування вставкою 110](#_Toc25867834)

[11.2 Сортування за ускладненими правилами 115](#_Toc25867835)

[11.3 Завдання для самостійної роботи 116](#_Toc25867836)

[11.4 Вимоги до звіту 117](#_Toc25867837)

[11.5 Контрольні питання 118](#_Toc25867838)

[*Рекомендована література* 118](#_Toc25867839)

[12 Обробка впорядкованих масивів 119](#_Toc25867840)

[12.1 Обробка упорядкованих масивів 119](#_Toc25867841)

[12.1.1 Пошук позиції елемента у впорядкованому масиві 119](#_Toc25867842)

[12.1.2 Вставка елементу до впорядкованого масиву 120](#_Toc25867843)

[12.1.3 Видалення елементу з упорядкованого масиву 122](#_Toc25867844)

[12.1.4 Злиття двох впорядкованих масивів 122](#_Toc25867845)

[12.2 Завдання для роботи 123](#_Toc25867846)

[12.3 Вимоги до звіту 123](#_Toc25867847)

[12.4 Контрольні питання 123](#_Toc25867848)

[13 структури 124](#_Toc25867849)

[13.1 Короткі теоретичні відомості 124](#_Toc25867850)

[13.1.1 Оголошення шаблону та ініціалізація структур 124](#_Toc25867851)

[13.1.2 Масиви структур 126](#_Toc25867852)

[13.1.3 Введення-виведення структур 127](#_Toc25867853)

[13.1.4 Сортування масивів структур 127](#_Toc25867854)

[13.2 Створення проекту «Результати атестації» 127](#_Toc25867855)

[13.2.1 Визначення глобальних типів даних програми 127](#_Toc25867856)

[13.2.1 Початкова ініціалізація масиву 128](#_Toc25867857)

[13.2.2 Інтерфейс користувача для проекту 128](#_Toc25867858)

[13.2.3 Функція відображення масиву на консолі 131](#_Toc25867859)

[13.2.4 Додавання нових даних до масиву структур 132](#_Toc25867860)

[13.2.5 Функція сортування масиву структур 134](#_Toc25867861)

[13.2.1 Реалізація сортування за ім’ям студента 134](#_Toc25867862)

[13.2.1 Реалізація сортування за результатами навчання 135](#_Toc25867863)

[13.2.2 Вибірка студентів, що мають середній бал не нижче заданого 135](#_Toc25867864)

[13.2.3 Виведення інформації про кількість студентів що мають заборгованості 136](#_Toc25867865)

[13.3 Завдання для самостійної роботи 137](#_Toc25867866)

[13.4 Вимоги до звіту 137](#_Toc25867867)

[13.5 Контрольні питання 137](#_Toc25867868)

[*Рекомендована література* 138](#_Toc25867869)

[14 файли 139](#_Toc25867870)

[14.1 Короткі теоретичні відомості 139](#_Toc25867871)

[14.1.1 Відкриття та закриття файлу 139](#_Toc25867872)

[14.1.2 Обробка текстових файлів 140](#_Toc25867873)

[14.1.1 Обмін блоками даних з файлом 141](#_Toc25867874)

[14.1.1 Робота з файлом в режимі прямого доступу 142](#_Toc25867875)

[14.2 Завдання для самостійної роботи 144](#_Toc25867876)

[14.3 Вимоги до звіту 145](#_Toc25867877)

[14.4 Контрольні питання 145](#_Toc25867878)

[*Рекомендована література* 145](#_Toc25867879)

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

Студент зобов’язаний до лабораторного заняття прочитати методичні вказівки до лабораторної роботи і спробувати виконати її самостійно. Під час лабораторного заняття студент показує викладачеві результати роботи, консультується з питань, що виникли, та завершує роботу. Обсяг виконаної роботи може бути різним, залежно від того, на яку оцінку претендує студент.

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 стандартів на оформлення технічної документації.

Окрім завдань, що вирішуються під час лабораторної роботи студент має виконати завдання для самостійної роботи. Результати виконання самостійної 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: звіт, проект, 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 програмуванняна С, С++» видання 2014 року.

# Знайомство з інтегрованим середовищем розробки “QT-creator”

Мета роботи:

* Отримати навички роботи з основними вікнами “QT-creator”.
* Створити найпростіший проект.

КОРОТКІ ВІДОМОСТІ ПРО “QT-creator”

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

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

ІСР «Qt creator» запускається шляхом вибору іконки «Qt creator» на робочому столі, або через меню «ПУСК» і підменю « Програми «.

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



Рисунок 1.1 – Стартове вікно ІСР «Qt creator»

### Головне меню середовища

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

### Панель режимів робіт

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

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

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

Режим дизайну використовується для створення графічних інтерфейсів користувача (GUI). З роботою у цьому режимі ми не будемо знайомитися, але студенти можуть вивчати цю тему самостійно, якщо буде бажання та час.

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

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

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

### Панель доступу до результатів опрацювання проекту

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

Консоль додатку відкривається автоматично, коли програма починає працювати.

### Створення проекту

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

Середовище «Qt creator» дозволяє створювати різні типи проектів, але поки що ми будемо створювати тільки проекти С++ у вигляді консольних застосувань.

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

Після вибору типу проекту слід дати цьому проекту ім’я та вибрати папку де проект буде зберігатися. Для імен папок та проекту слід використовувати латинський шрифт. Доречно зберігати усі проекти в одній папці та давати проектам зрозумілі імена. Краще створювати свою персональну папку, а не використовувати папку, що створюється за замовчуванням.

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

### Панель редагування програм

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



Рисунок 1.2 – Панель редагування програм

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

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

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

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

Перш за все напишемо ще дві директиви #include такого змісту:

 На рисунку 1.2 показано місце, де слід написати ці рядки.

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

За допомогою директиви #include <stdio.h> ми підключаємо заголовний файл бібліотеки С для потокового введення виведення інформації. Директива #include <iostream> означає, що ми підключаємо заголовний файл аналогічної бібліотеки С++.

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



Тепер ми маємо можливість спілкуватися з програмою через консоль.

Попросимо програму вивести на консоль привітання.

Для цього достатньо дописати у файл main.cpp ще два рядки:



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

У першому рядку для виведення послідовності символів використовується функція puts(), у другому те саме завдання виконує об’єкт С++ cout .

У подвійних лапках наводиться текст, що підлягає виведенню. Символи «\n» означають, що після виведення тексту слід перейти до нового рядка. А операція << означає, що текст «Hello from C++! \n» буде передано об’єкту cout.

Спробуйте виконати цю програму.

Автоматичне вирівнювання тексту програми

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

* кожний оператор має займати окремий рядок;
* закриваюча фігурна дужка для функції розташовується в окремому рядку;
* оператори функції розміщуються з відступом від фігурних дужок;
* пробіли не ставляться з обох сторін круглих дужок, що йдуть за круглими дужками.

За допомогою контекстного меню панелі редагування (викликається правою кнопкою миші) можна вирівняти фрагмент програми (авто відступ), якщо виділити цей фрагмент.

Спробуйте зробити це, попередньо розташувавши рядки нерівно.

Коментарі до тексту програми

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

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

У мові С++ коментарі позначаються двома похилими рисками.

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

Спробуйте зробити це.

Повернення у режим редагування

Клавіша Esc забезпечує повернення до редактора із будь якого вікна.

### Функція main()

Програма на С або С++ являє собою сукупність різноманітних оголошень та функцій.

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

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

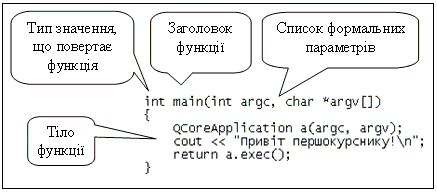


Рисунок 1.3 - Структура функції main()

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

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

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

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

int main()

{

…

return 0;

}

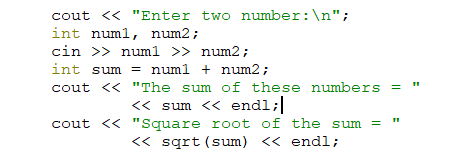
У функції main(), що наведена на рисунку 1.3 і створюється разом із новим проектом дещо складніший вигляд. Тіло містить описи та оператори, що забезпечують відкриття вікна консолі.

Ви можете використовувати і спрощений вигляд функції, вилучивши усе те, що було створено автоматично, але у цьому випадку доведеться вирішувати проблему, як зберегти на екрані вікно консолі після завершення роботи функції main(). Тому, краще нічого не видаляти, а тільки доповнювати зразок.

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

### Приклад програми з введенням числових даних

Створіть новий проект і додайте до функції main() такий код:



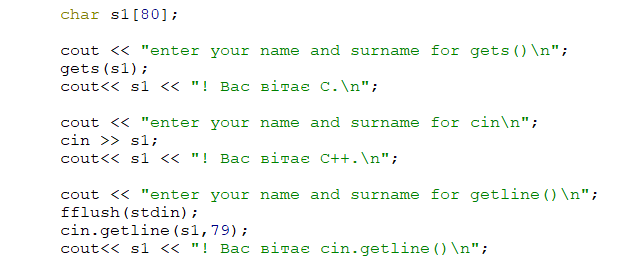
Не забутьте додати директиву #include (подумайте, яку саме) та оголошення using. Додайте також директиву #include <math.h>, яка надає доступ до бібліотеки математичних функцій.

У другому рядку цього коду оголошуються дві змінні – num1 та num2, які призначені для зберігання цілих чисел (тип int) .

У третьому рядку об’єкт С++ cin використовується для введення з консолі чисел у змінні num1 та num2. Числа при введенні розділяються пробілом, або переходом на наступний рядок.

### Приклад програми з введенням символьних даних

Створіть новий проект і додайте до функції main() такий код:



Не забутьте додати директиви #include та оголошення using.

У першому рядку цього коду оголошується рядок символів s1.

У рядок s1 буде вводитися ім’я та прізвище користувача за допомогою функції gets(), об’єкту cin та методу getline() цього об’єкту.

Запуск цієї програми має продемонструвати, що об’єкт cin вводить символьні дані до першого пробілу. Більше того, він не очищує після введення буфер вводу.

А метод getline об’єкт cin дозволяє вводити символьні рядки, що містять пробіли. До того ж ця функція дозволяє обмежити кількість символів, що буде прочитано.

Функція fflush(stdin) очищає буфер вводу. Спробуйте закоментувати цей рядок і подивитися результат виконання.

### Дослідження специфічних констант

Константи цілих чисел

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

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

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

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

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

Символьні константи

Символьні константи поділяються на графічні символи, керуючі слеш-символи та ескейп-послідовності. У будь якому варіанті символьні константи розташовують між одинарними лапками. Наведемо приклади символьних констант: ‘a’ (літера а), ‘\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.

Константи дійсних типів

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

Створіть новий проект і додайте до функції main такий код.

cout << «022 = « << 022 << endl;

cout << «0x1b = « << 0x1b << endl;

cout << «0b101 = « << 0b101 << endl;

cout << «'\\a' = « << '\a' << endl;

cout << «2e3 = « << 2e3 << endl;

cout << «2e-1 = « << 20e-1 << endl;

cout << «5 = « << bitset<32>(5).to\_string() << endl;

cout << «-5 = « << bitset<32>(-5).to\_string() << endl;

cout << «255 = « << hex << 255 <<endl;

Не забутьте додати необхідні директиви, а також директиву   
#include <bitset>

Остання директива дозволяє використовувати функції, за допомогою яких ціле число можна відобразити так, як воно зберігається в пам’яті.

### Збереження проекту

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

Якщо проект треба буде розгорнути на іншому комп’ютері, то можна скопіювати папку з ім’ям проекту і відкрити проект там де потрібно, звернувшись до файлу з розширенням .pro.

Можна скопіювати з папки, що має ім’я проекту лише файли main.cpp та файл з розширенням .pro.

Нарешті, можна зберегти тільки файл main.cpp, а на іншому комп’ютері створити новий проект і у ньому замінити текст файлу main.cpp.

Завдання для самостійної роботи

Опрацювати розділ 1 підручника [1] і записати у звіті відповіді на контрольні питання.

Опрацювати розділ 2 підручника [1].

Виконати пункти 9 і 10 завдань до самоконтролю розділу 2 підручника.

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* Відповіді на контрольні питання розділу 1 підручника.
* Тексти усіх програм, що створювалися, з коментарями і результатами тестування у вигляді копій консолі.
* Висновки.

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 перенести проект на інший комп’ютер.

## *Рекомендована література*

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

# Типи даних та розрахунки за формулами

Мета роботи:

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

Короткі теоретичні відомості

### Змінні і константи

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

### Типи даних

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

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

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

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

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

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

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

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

Ці типи можна поділити на такі:

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

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

### 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.

Основні арифметичні типи

Основними типами є такі:

char – тип для символів;

int – тип для цілих чисел;

float – тип для дійсних чисел;

double – дійсні числа подвійної точності;

bool – логічний тип, що може приймати значення 0, або 1.

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

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

Модифіковані арифметичні типи

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

|  |  |  |
| --- | --- | --- |
| Таблиця 2.1 – Модифікатори для типів мови С++ | | |
| Модифікатор | Значення | Застосовують до типів |
| signed | із знаком | char, int |
| unsigned | без знаку | char, int |
| long | довгий | int, long int, double |
| short | короткий | int |

Як бачимо, використання модифікаторів обмежено, за виключенням типу int, до якого можна застосувати будь який модифікатор. Та на практиці область використання ще вужча. Модифікатор signed практичного сенсу не має, бо за принципом замовчування усі типи мають знак. Для знаку виділяється старший біт коду числа. Якщо значення цього біту нуль, то число додатне, якщо одиниця - то від’ємне.

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

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

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.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця 2.2 – Скорочені назви та граничні константи для типів С++ | | | |
| Повна назва типу | Скорочена назва | Тип Qt | Граничні константи |
| char | char | [qint8](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qint8-typedef) | CHAR\_MIN |
| char | char | [qint8](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qint8-typedef) | CHAR\_MAX |
| unsigned char | uchar | [quint8](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#quint8-typedef) | UCHAR \_MAX |
| short | short | [qint16](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qint16-typedef) | SHRT\_MIN |
| short | short | [qint16](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qint16-typedef) | SHRT\_MAX |
| unsigned short | ushort | [quint16](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#quint16-typedef) | USHRT\_MAX |
| int | int | [qint32](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qint32-typedef) | INT\_MIN |
| int | int | [qint32](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qint32-typedef) | INT\_MAX |
| unsigned int | uint | [quint32](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#quint32-typedef) | UINT\_MAX |
| long int | long | [qint64](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qint64-typedef) | LONG\_MIN |
| long int | long | [qint64](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qint64-typedef) | LONG\_MAX |
| unsigned long int | ulong | [quint64](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#quint64-typedef) | ULONG\_MAX |
| long long int | long long | [qlonglong](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qlonglong-typedef) | LLONG\_MIN |
| long long int | long long | [qlonglong](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qlonglong-typedef) | LLONG\_MAX |
| unsigned long long int | ? | [qulonglong](http://doc.crossplatform.ru/qt/4.4.3/qtglobal.html#qulonglong-typedef) | ULONG\_LONG\_MAX |
| float |  |  | FLT\_MIN |
| float |  |  | FLT\_MAX |
| double |  |  | DBL\_MIN |
| double |  |  | DBL\_MAX |
| long double |  |  | DBL\_MAX |

Переліки

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

Оголошення переліку має такий вигляд:

enum <імя переліку> {<список іменованих константз>};

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

enum consoleColors{

BLACK, BLUE, GREN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY, LIGHTBLUE,

LIGHTGREN, LIGHTCYAN, LIGHTRED, LIGHTMAGENTA, YELLOW, WHITE

};

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

У разі потреби елементам можна присвоювати інші значення.

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.

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).

### Описи змінних

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

У простому випадку інструкція оголошення змінної виглядає так:

<*тип*> <*имя*>;

де: <ім’я> - ім’я змінної;

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

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

У розділі може бути і декілька інструкцій оголошення, наприклад:

int i, j, k ;

float x;

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

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

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

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

const double twoPi=2\*3.1425926;

Макроконстанти

У мові С++ 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)

### Операція розміру sizeof

Операція sizeof повертає обсяг пам’яті, яку займає або потребує операнд цієї операції, тобто розмір операнду. 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 імя змінної

sizeof (вираз)

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

### 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 |

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.

### Операції присвоєння

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.

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.

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 х має значення 5. Тоді значення виразу (3+ ++х) буде дорівнювати 9, а значення виразу (3+х++) буде дорівнювати 8, хоча «х» у обох випадках отримає значення 6.

### 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 |
|  | (a+b)/(a-b)/x |
|  | sqrt(pow (sin(x),2)+b) |
|  | exp(2.5\*x) |
|  | 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.

### Пріоритети операцій у С++

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

|  |  |
| --- | --- |
| Таблиця 2.5 – Пріоритети операцій С++ | |
| 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 |

### Узгодження типів

У мові С++ 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.

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.

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.

Явне перетворення типів

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

(*тип*) *вираз*

*тип* (*вираз*)

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

### Бібліотека математичних функцій cmath

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

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

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

У таблиці 2.6 наведено деякі функції бібліотеки cmath.

|  |  |
| --- | --- |
| Таблиця 2.6 Функції бібліотеки cmath | |
| Приклад звернення до функції | Що обчислює функція |
| fabs(вираз) | Модуль числа |
| sqrt(вираз) | Квадратний корінь |
| pow(вираз, ступінь) | Піднесення до ступеню |
| exp(вираз для степеню числа e) | Експонента |
| log(вираз) | Натуральний логарифм |
| log10(вираз) | Логарифм за основою 10 |
| sin(вираз) | Синус |
| cos(вираз) | Косинус |
| tan(вираз) | Тангенс |
| asin(вираз) | Арксинус |
| acos(вираз) | Арккосинус |
| atan(вираз) | Арктангенс |
| [sinh](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cmath-math-h/funkciya-sinh/)(вираз) | Гіперболічний синус |
| [cosh](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cmath-math-h/funkciya-cosh/)(вираз) | Гіперболічний косинус |
| [tanh](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-cmath-math-h/funkciya-tanh/)(вираз) | Гіперболічний тангенс |
| ceil(вираз) | Округлення до найближчого більшого |
| floor(вираз) | Округлення до найближчого меншого |
| fmod(вираз ділене, вираз дільник) | Залишок від ділення з плаваючою точкою |

### 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, у 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.

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 такогот рядка коду

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

на консолі з’явиться такий текст

«Сума чисел 10 та 5 дорівнює 15».

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

У таблиці 2.1 наведено деякі коди форматування наведені.

|  |  |
| --- | --- |
| Таблиця 2.1 – Коди форматування функції printf() | |
| Код | Аргумент |
| %с | Символ типу char |
| %d | Десяткове число цілого типу зі знаком |
| %lld | Десяткове число цілого типу long long зі знаком |
| %е | Наукова нотація (е нижнього регістру) |
| %Е | Наукова нотація (Е верхнього регістру) |
| %f | Десяткове число з плаваючою точкою |
| %g | Використовує код% е або% f - той з них, який коротше |
| %G | Використовує код% Е або% f - той з них, який коротше |
| %о | Вісімкове ціле число без знаку |
| %s | Рядок символів |
| %u | Десяткове число цілого типу без знаку |
| %llu | Десяткове число цілого типу long long без знаку |
| %х | Шістнадцяткове ціле число без знаку (літери нижнього регістру) |
| %Х | Шістнадцяткове ціле число без знаку (літери верхнього регістру) |
| %llх , %llХ | Те саме для типу long long |
| %р | Покажчик на якийсь елемент |
| %n | Покажчик на змінну цілого типу, в яку буде занесено кількість надрукованих символів |
| %% | Вивести знак % |

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.

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, о, u, х і X. Специфікатор l «говорить» printf () про те, що далі йдуть дані типу long. Наприклад,% ld означає, що потрібно вивести long int. Специфікатор h вказує printf (), що потрібно відобразити short int. Отже,% hu вказує, що дані мають тип short unsigned int.

Модифікатор l може бути також пред специфікаторами типу з плаваючою точкою е, Е, 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, е або Е, то десяткова точка буде ставитися навіть за відсутності цифр після коми. Якщо поставити # перед специфікатором формату х, то шістнадцяткове число буде виведено з префіксом 0х. Використання # зі специфікатором о 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 [1].

### Форматне виведення даних через cout

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

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

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

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

cout << «Науковий формат i знак + :\n»;

cout.setf (ios::showpos);

cout.setf (ios::scientific);

cout <<123 << « « << 123.23<<» « << 123 <<»\n»;

////////////////////////////////////////////////////////////////////////

cout<<»10 значущих цифр пiсля коми:\n»;

cout.precision (10);

cout<<1/3.0<<» « <<0.1<<»\n»;

/////////////////////////////////////////////////////////////////////////

cout<<»Ширина поля 10 знакiв:\n»;

cout.width(10);

cout << 123 <<» «<< 123.23<<» «<<123 << «\n»;

///////////////////////////////////////////////////////////////////////////

cout<<»Використання символiв заповнювачiв:\n»;

cout.fill (‘#’);

cout.width(10);

cout <<123<< « « << 123.3<< « «<<123<< «\n»;

Завдання на лабораторну роботу

Опрацювати розділи 3 та 4 з підручника.

Знайти відповіді на питання 1-12 розділу 3 підручника [1], стр.32.

Написати програму, яка буде послідовно вирішувати такі завдання:

* видавала на консоль результат виконання усіх арифметичних операцій для введеної пари цілих чисел;
* виводила результати обчислення за формулами питання 7 розділу 4 підручника, сторінка 65, для значень змінних, що вводить користувач;
* видавала на консоль розмір основних типів даних та їх граничних значень у вигляді відформатованої таблички;
* виводила результати виконання наступного коду.

cout << «'F' + 1 = « << 'F' + 1 << endl;

cout << «true + 2 = « << true + 2 << endl;

cout << «3 & 5 = « << (3 & 5) << endl;

cout << «3 | 5 = « << (3 | 5) << endl;

cout << «3 ^ 5 = « << (3 ^ 5) << endl;

cout << «3 << 1 = « << (3 << 1) << endl;

cout << «-3 << 1 = « << (-3 << 1) << endl;

cout << «3 >> 1 = « << (3 >> 1) << endl;

cout << «-5 = « << bitset<8>(-5).to\_string() << endl;

cout << «-5 >> 1 = « << bitset<8>(-5 >> 1).to\_string();

Ви маєте проаналізувати отримані результати і пояснити, чому саме такими вони є.

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* Відповіді на питання з розділу 3 підручника
* Тексти програми з коментарями
* Результати виконання програм у вигляді копій консолі.
* Висновки.

Контрольні питання

* [Змінні і константи.](#_Toc347906362)
* [Тип даних.](#_Toc347906363)
* [Арифметичні типи даних.](#_Toc347906364)
* [Описи змінних.](#_Toc347906365)
* [Операція розміру sizeof](#_Toc347906366).
* [Арифметичні операції.](#_Toc347906367)
* [Операції присвоєння.](#_Toc347906368)
* Побітові операції
* [Вирази.](#_Toc347906369)
* [Пріоритети операцій у С++.](#_Toc347906370)
* [Узгодження типів.](#_Toc347906371)
* [Бібліотека математичних функцій cmath.](#_Toc347906372)
* [Використання printf() для виведення даних.](#_Toc347906373)

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

# Функції

Мета роботи:

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

Короткі теоретичні відомості

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

### Правила написання функцій

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

*nип\_значення\_що\_повертається* *ім’я\_функції*(*список\_формальних\_параметрів*)

{

*тіло\_функції*

}

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

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

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

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

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

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

return вираз ;

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

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

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

float sumTwoNumber (float a, float b)

{float z;

z=a+b;

return z;

}

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

Ту саму функцію можна записати і коротше:

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

### Виклик функції

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

ім’я\_функції(список\_фактичних\_параметрів)

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

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

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

…  
 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 у. Фактичними параметрами у цьому випадку є константи 4.2 та 5.4, а результатом число 9.6. Друге звернення до нашої функції відбувається у виразі, що передається об’єкту cout. Фактичними параметрами у цьому випадку є змінна х та вираз у/2, а результатом буде число 5.5.

### Прототип функції

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

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

*тип\_значення\_ що\_повертається ім’я\_функції*(*список*\_*формальних\_параметрів*);

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

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

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

float sumTwoNumber (float a, float b);

Той же прототип може виглядати і так:

float sumTwoNumber (float, float);

### 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 namespec std;

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

### 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 &.

Передача параметрів за значенням

Передача параметрів за значенням передбачає, що під час виклику функції у пам’яті буде виділена спеціальна область для запису копій значень фактичних параметрів, з якими і буде працювати функція.

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

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

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

Передача параметрів через посилання

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

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

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

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

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

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

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

### Області оголошення та доступу до імен

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

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

Глобальні та локальні змінні

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

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

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

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

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

Локальні змінні створюються тільки після виклику функції та існують тільки до завершення роботи функції, або блоку. Пам’ять для цих змінних автоматично виділяється, а потім звільнюється у процесі виконання програми. Тому такі змінні ще називають автоматичними.

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.

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. На перший погляд тут складається враження, що ніякої випадковості нема, але слід враховувати, що константа «а» дуже велика. Результат множення змінної «х» на таке велике число майже завжди буде перевищувати максимально можливе значення для типу unsigned long, внаслідок чого старші біти результату будуть втрачатися. Таким чином результат стає ніби то випадковим.

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

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

Початкове значення змінної «х» 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 «х» оголошена із специфікатором **statsc**.

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

### Макроси з параметрами

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

Синтаксис макросу з параметрами такий:

#define ім’я\_макросу(список\_ параметрів) вираз\_для\_заміни

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

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

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

Звернення до цього макросу у програмі може виглядати так:

cout << POW( 3 + x , 4.5 ) << “\n”

Реализація проекту «function»

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

|  |  |
| --- | --- |
| Таблиця 3.1 – Завдання для проекту | |
| Варіант | Формула |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |

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

 (3.1)

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

### Підключення допоміжних файлів

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



Рисунок 3.1 – Директиви #include програмного файлу

Файл iostream забезпечує реалізацію обміну інформацією з консоллю.

Файл cmath забезпечує доступ до математичних функцій.

### Створення макросу з параметрами

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



Рисунок 3.2 – Визначення макросу з параметрами

### Створення прототипів функцій

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



Рисунок 3.3 – Прототипи функцій програмного файлу

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

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

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

### Функція main() програмного файлу

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

Код цієї функції наведено на рисунку 3.4.

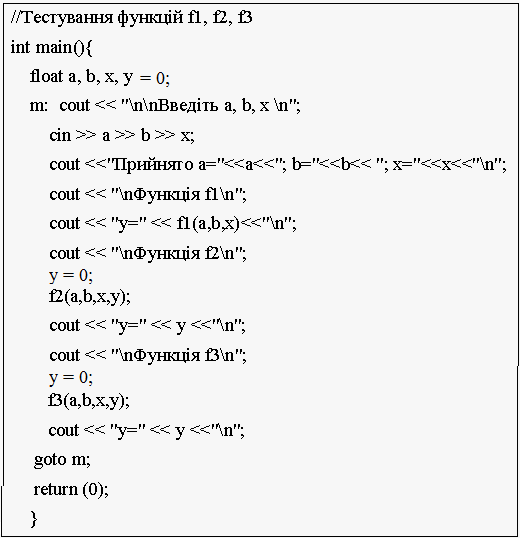


Рисунок 3.4 – Функція main() програмного файлу

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

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

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

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

f2(a, b, x, y);

Після цього отриманий результат можна вивести на консоль:

cout << “y = ” << y << “\n”;

### Функція f1(), що повертає значення

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

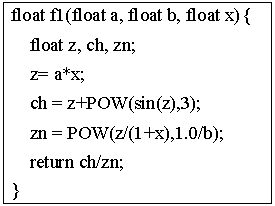


Рисунок 3.5 – Функція, що повертає значення

### Функція f2() типу void

Ця функція безпосередньо результат не повертає, але для збереження результату використовується посилання &y на змінну з формальним ім’ям «у». Саме цій змінній і присвоюється результат обчислень. Код функції f2() наведено на рисунку 3.6.

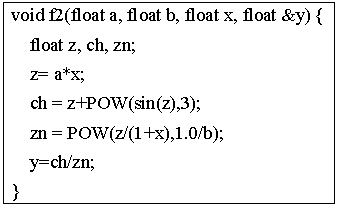


Рисунок 3.6 – Функція, що повертає значення через параметр

### Функція з виведенням проміжних результатів

Код функції f3() наведено на рисунку 3.7.

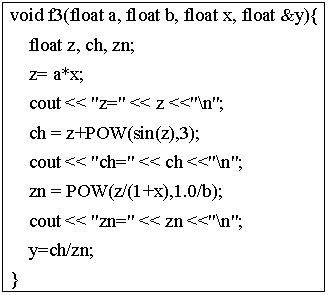


Рисунок 3.7 – Функція з виведенням проміжних результатів

Дослідження створеного проекту

### Дослідження передачі параметрів за посиланням

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

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

Відновіть заголовок процедури.

Зробіть висновки і зафіксуйте результати досліджень у звіті.

### Анализ передачи параметров по значению

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

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

Відновіть текст функції.

Зробіть висновок і зафіксуйте в звіті.

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* «Завдання на лабораторну роботу», де має бути посилання на формулу, за якою виконувалися розрахунки (можна скопіювати і вставити як рисунок).
* «Контрольний розрахунок». Тут треба для контрольного набору вхідних даних виконати розрахунки  проміжних і кінцевого результатів за допомогою калькулятора, щоб можна було перевірити результат роботи програми.
* Тексти функцій програми з коментарями і результати тестування проекту у вигляді копій консолі. Кожний варіант функції окремим підрозділом.
* Результати дослідження способів передачі параметрів до функції
* Висновки.

Контрольні питання

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

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

# Логічний тип даних і розгалуження у програмах

Мета роботи:

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

Короткі теоретичні відомостіІ

### Логічний тип даних

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

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

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

|  |  |
| --- | --- |
| Таблиця 4.1 - Операції порівняння | |
| Назва операції порівняння | Запис на мові С |
| Менше | < |
| Менше або дорівнює | <= |
| Більше | > |
| Більше або дорівнює | >= |
| Дорівнює | == |
| Не дорівнює | != |

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 |

Логічні вирази

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

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

.

|  |  |
| --- | --- |
| Таблиця 4.3 – Пріоритети операцій | |
| Операції | Пріоритет |
| \* / % | 3 |
| + - | 4 |
| << >> | 5 |
| < > <= >= | 6 |
| = = != | 7 |
| & ^ | && || | 8,9,10,11,12 |

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

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

### Алгоритми з розгалуженнями

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

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

Деякі умовні позначення, що використовуються при складанні схем алгоритмів, наведені в таблиці 4.4. При зображенні алгоритму окремі блоки нумеруються і з’єднуються лініями. Стрілки використовуються тільки для зворотних напрямків (вгору і ліворуч).

Таблиця 4.4 - Умовні позначення для схем алгоритмів

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

В якості прикладу розглянемо схеми алгоритму розв’язання квадратного рівняння ax2 + 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 4.1- Схема алгоритму аналізу коефіцієнтів квадратного рівняння

На рисунку 4.2 зображена схема алгоритму рішення лінійного рівняння. У цьому алгоритмі аналізуються значення решти коефіцієнтів. Якщо обидва вони дорівнюють нулю, то рівняння 0х + 0 = 0 задовольняє будь-яке значення х. 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 х =-c / b.



Рисунок 4.2 – Схема алгоритму виводу результатів  
розв’язку лінійного рівняння

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

### Програмування розгалужень

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.



Рисунок 4.3– 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 4.4.

void testCoef(double a, double b, double c) {

cout << “a = a” << a << “, b = “ << b << “, c = ” << c <<endl;

if (a==0)

// Звернення до функції рішення лінійного рівняння

lin\_riv(b,c);

else

// Звернення до функції рішення квадратного рівняння

kv\_riv(a,b,c);

}

Рисунок 4.4 – Програмна реалізація алгоритму аналізу коефіцієнтів квадратного рівняння

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

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



Рисунок 4.5 – Скорочена форма оператора **if**

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

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



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

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



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

Умовна операція

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



Рисунок 4.8 – Синтаксис теренарного оператора

Тут умова - це вираз, результат якого приводиться до логічного типу. Якщо результат обчислення умови приймає значення “true”, то результатом операції буде значення виразу 1. Якщо результат обчислення умови приймає значення “false”, то результатом операції буде значення виразу 2. На рисунку 4.9 наведено приклад використання вкладеного теренарного оператора у функції для розрахунку стипендії.

.

Рисунок 4.9– Приклад використання теренарного оператора

Оператор вибору swith

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 4.10.

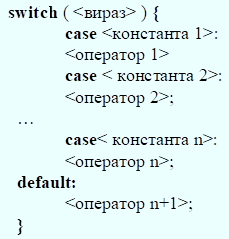


Рисунок 4.10 – Синтаксис оператора **switch**

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 defaul, якщо ця частина присутня.

В якості прикладу (рисунок 4.11) розглянемо функцію, що перераховує значення змінної х в залежності від значення параметру і.

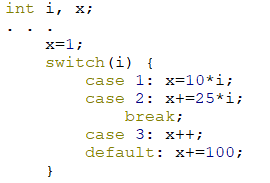


Рисунок 4.11 – Приклад використання оператора **switch**

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

Якщо і = 2, то результатом буде х = 51, тому що спрацює тільки другий варіант.

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

Якщо і = 4, то результатом буде х = 101, тому що спрацює тільки дефолтний варіант.

### Оператор переходу goto

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



Рисунок 4.12– Синтаксис оператора **goto**

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

Реализація проекту «if\_switch»

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

### Початковий інтерфейс проекту

Після запуску програми на консолі має з’явитися перелік результатів тестування можливих варіантів роботи алгоритму і пропозиція користувачеві (prompt) ввести свій варіант даних. На рисунку 4.13 наведено можливий вигляд консолі після запуску програми.

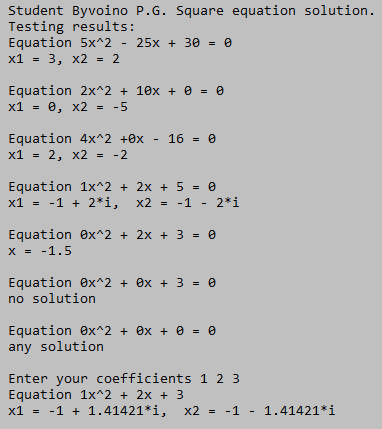


Рисунок 4.13 – Вигляд консолі після запуску програми

### Структура проекту

Проект має складатися з функції main і допоміжних функцій, перелік яких наведено нижче:

void testCoef(double a, double b, double c);

void printLinEqResult(double b, double c);

double calcLinRoot(double b, double c);

void printSqrEqResult(double a, double b, double c);

int calcSqrData(double a, double b, double c,

double &re, double &im);

Функція testCoef() 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 х = -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 printSqrEqResult() виводить результат розв’язання квадратного рівняння. 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 значення кореня потрібно мати дві складових. Ці складові обчислює і повертає через параметри функція calcSqrData(). Окрім того, функція через return повертає результат обчислення виразу d>=0, що дозволяє перевірити знак дискриминанту d.

У випадку, коли дискримінант d від’ємний, перша складова, яка обчислюється за формулою re = -b/2/a, являє собою дійсну частину кореня, а друга – уявну і обчислюється за формулою im = sqrt(abs(d)/2/a.

Коли d>=0, то перший корінь рівння є сумою цих складових x1=re+im, а другий – різницею, x2 = re – im.

Схему алгоритму цієї функції можна знайти у презентації лекції.

Функція calcSqrData(), як вже було сказано, обчислює складові коренів рівняння і повертає «знак» дискриминанта (d >= 0).

### Функція main() програмного файлу

Функція main() програмного файлу 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.

Тестування реалізується викликом функції testCoef() із заданим набором коефіцієнтів. Наприклад, варіант тестування комплексних коренів може виглядати так:

cout << «\n Equation 1x^2 + 2x + 5 = 0\n»;

testCoef(1, 2, 5);

### Завдання із підручника

Створити проект для реалізації завдання 11 розділу 6 підручника [1], сторінка 111. Обмін значеннями змінних реалізувати через виклик функції, куди змінні передаються, як параметри. Введення даних і виведення результатів має бути у функції main().

Вимоги до звіту

Звітом з роботи є:

* файл main.cpp в якому спочатку тестується функція обміну чисел завдання з підручника, після чого викликаються тести для квадратного рівняння. У початковому коментарі файлу має бути вказано тему проекту, прізвище та ініціали студента та його групу. До кожної з функцій ОБОВ’ЯЗКОВО має бути коментар про її призначення, вхідні параметри, та що вона і яким чином повертає. Код має бути ОБОВ’ЯЗКОВО відформатований. Файли без коментарів або невідформатовані не розглядаються.
* файл копії консолі з результатами роботи проекту. Колір консолі має бути світлим. Текст темним. Розмір консолі має відповідати обсягу даних. Текст зручним для читання.

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.

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

# Використання циклів „while” і „do...while” в Ітераційних алгоритмах

Мета роботи:

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

Короткі теоретичні відомості

### 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.

### Оператор while

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

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



Рисунок 5.1 – Синтаксис оператора **while**

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

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

* обчислюється значення виразу <умова виконання>;
* якщо значення виразу <умова виконання > дорівнює **false** або 0, тобто умова не виконується, виконання <операторів тіла циклу > припиняється;.
* якщо значення виразу <умова виконання дорівнює **true** або не 0 (умова виконується), то виконуються <оператора тіла циклу >, розташовані між фігурними дужками;
* після цього знову все повторюється.

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

На рисунку 5.2 представлено ​​схему алгоритму виконання цього циклу.

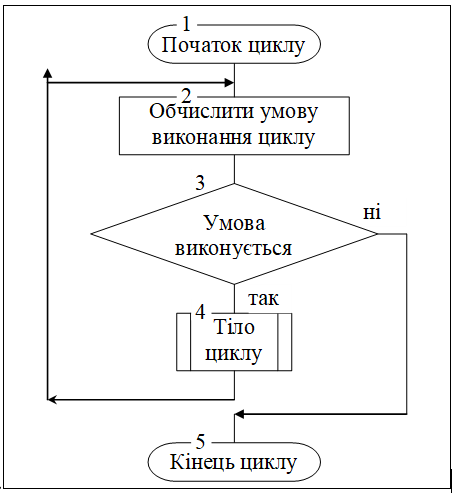


Рисунок 5.2 – Схема алгоритму виконання циклу while

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

Текст цієї функції наведено нижче:

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

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

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

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

### 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):



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

Текст програми наведено після схеми алгоритму.

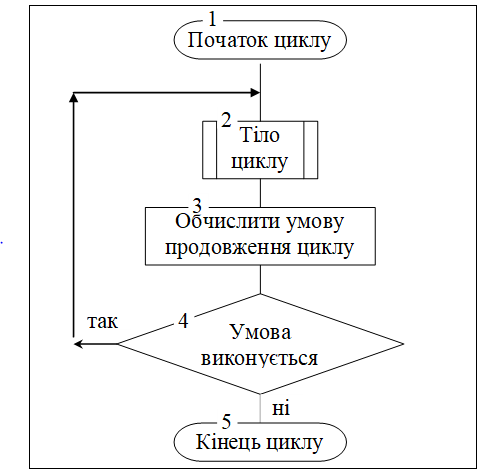


Рисунок 5.4 – Схема алгоритму виконання циклу **do…while**

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

Текст цієї програми наведено нижче.

string p=»qwerty»;  
 string inpt;  
 do{  
 cout<<»/nВведiть пароль «;  
 cin>>inpt;  
 }while(p!=inpt);  
 cout<<»OK!\n»;  
 …

### Переривання циклу

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

На відміну від break оператор continue перериває виконання тільки тіла циклу і одразу ж переходить до обчислення умови продовження циклу.

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

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

### 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 для самостійної роботи 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 х0 (початкове наближення) і підставляється до ітераційної формули. Отримане за ітераційною формулою значення розв’язку порівнюється з попереднім. Якщо ці значення істотно відрізняються один від одного, то нове наближене значення підставляється в ітераційну формулу замість старого і на його основі отримують нове наближене значення. Так триває доти, поки нове і старе наближення стануть достатньо близькими один до одного.

Приклад ітераційного алгоритму для обчислення кубічного кореня

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

 (5.1)

де а – число, з якого добувається корінь, xold – попереднє наближене значення кореня, xnew – наступне, більш точне, ніж xold значення кореня.

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

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

В алгоритмі передбачено переривання циклу за допомогою оператора break в тому випадку, якщо змінна Xold дорівнює нулю, для того, щоб виключити ділення на 0. Але це не означає, що таке треба робити у кожному з ваших завдань.

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



Рисунок 5.5 – Схема алгоритму знаходження кубічного кореня

### Функції для обчислення кубічного кореня

На рисунку 5.6 наведено текст функції, що повертає результат обчислення кубічного кореня, яка реалізована відповідно до схеми алгоритму 5.5.

На рисунку 5.6 наведено текст функції, яка обчислює кубічний корінь за тим же алгоритмом, але результат у вигляді послідовних наближень до значення кореня та результат виводить на консоль. Окрім того, на консоль виводиться результат обчислення стандартним способом для порівняння

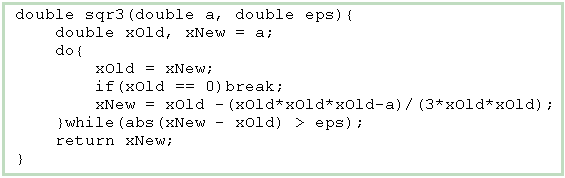


Рисунок 5.5 – Функція обчислення кубічного кореня

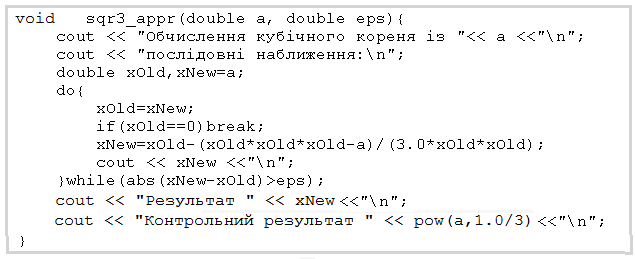


Рисунок 5.6 – Функція для обчислення кореня кубічного з виведенням послідовних наближень до кореня

Завдання для самостійної роботи

1. У межах виконання лабораторної роботи студент повинен самостійно розробити алгоритми та програми розв’язання рівняння ітераційним методом у відповідності зі своїм варіантом завдання з таблиці 5.1 Номер варіанта вибирається відповідно до останньої цифри залікової книжки. **Потрібно реалізувати дві функції.** Перша має тільки повертати знайдений корінь. Друга має виводити на консоль послідовні наближення.

2. Написати функцію, що реалізує завдання 14 розділу 6 підручника, сторінка 111.

Для тестування функцій написати метод main. У цьому методі має бути перевірка вхідних даних на коректність (обмеження у завданні), а також перевірка коректності знайденого результату за функцією 1. Для цього знайдений корінь слід підставити у рівняння, що вирішується. Зрозуміло, що результат обчислення у цьому випадку має бути близьким до нуля.

Таблиця 5.1 Рівняння, які вирішуються методом ітерацій

|  |  |  |  |
| --- | --- | --- | --- |
| Варіант | Рівняння | Ітераційна формула | Обмеження |
| 1 | 2 | 3 | 4 |
| 0 |  |  | 0<ab<1, x0>0 |
| 1 |  |  | a>0, b>0,  b/a<6, x0<π/2 |
| 2 |  |  | a>0, b>0,  c>0, x0>0 |
| 3 |  |  | a>1, 0< b<1,  x0>0 |
|  | | | |
| Продовження таблиці 5.1 | | | |
| 1 | 2 | 3 | 4 |
| 4 |  |  | ab>c,  |x0|<π/(2b) |
| 5 |  |  | a>0, b>0  c>1, x0>0 |
| 6 |  |  | a>0, c>0,  x0>0 |
| 7 |  |  | a>1, c>1,  b<c, x0>1 |
| 8 |  |  | a>0, b>0,  c>0, x0>0 |
| 9 |  |  | a>0, b>0,  c>0, x0>0 |

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* Умови індивідуальних завдань та схеми алгоритмів їх вирішення.
* Тексти функцій для індивідуальних завдань з коментарями.
* Результати тестування проекту у вигляді копій консолі.
* Висновки.

Контрольні питання

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

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

# Використання циклів „while” і „do...while” для обчислення сум нескінченних рядів

Мета роботи:

* Познайомитися з особливостями використання циклів **while** та **do…while** для обчислення сум нескінченних послідовностей
* Створити проект, що забезпечує обчислення сум нескінченного ряду.

### 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.

В якості прикладу обчислення суми рядурозглянемо алгоритм обчислення синуса деякого числа. Синус можна представити як суму нескінченного ряду 6.1.

 (6.1)

Особливість даного ряду полягає в тому, що кожен член ряду, починаючи з другого, може бути знайдений з попереднього за формулою 6.2.

 (6.2)

Алгоритм обчислення синуса за формулою 6.1 з використанням співвідношення 6.2 представлено на рисунку 6.1.

Слід мати на увазі, що ряд 6.1 при великих значеннях х сходиться повільно, а факторіал і «х в ступеню» ростуть дуже швидко. Це призводить до того, що значущі цифри цих чисел перестають міститися в розрядній сітці і, отже, обрізаються, внаслідок чого результат спотворюється.

Для усунення цього недоліку в розглянутий алгоритм перед основним циклом накопичення суми ряду варто було б додати додатковий цикл. У цьому циклі змінна х повинна зменшуватися кожен раз на величину періоду синуса, який дорівнює 2π. Цикл працює доти, поки х не стане менше, ніж 2π.



Рисунок 6.1 – Алгоритм обчислення синуса

### Функції для обчислення синуса

На рисунку 6.2 наведено текст функції для обчислення синусу, яка реалізована відповідно до схеми алгоритму 6.1. Ця функція повертає значення синусу.

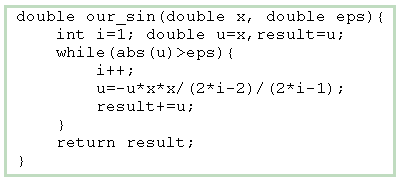


Рисунок 6.2 – Функція для обчислення синуса

На рисунку 6.3 наведено текст функції, яка обчислює синус за тим же алгоритмом, але виводить результат на консоль а також послідовні наближення до результату. Окрім того, на консоль виводиться результат обчислення синусу стандартним способом для порівняння.

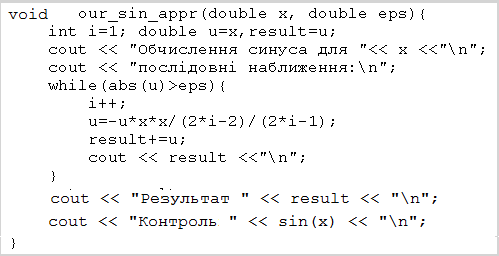


Рисунок 6.3 – Функція для обчислення синуса з виведенням послідовних значень суми ряду

Завдання для самостійної роботи

У межах виконання лабораторної роботи студент повинен самостійно розробити алгоритми та програми обчислення суми нескінечного ряду відповідно варіанту з таблиці 6.1. Номер варіанта вибирається відповідно до останньої цифри залікової книжки.

Потрібно реалізувати дві функції. Перша має тільки повертати суму ряду. Друга має виводити на консоль послідовні наближення, а також результат порівняння знайденого результату з результатом, що знайдено за допомогою звичайних обчислень, чи за допомогою стандартних функцій.

Окрім того слід реалізувати завдання 18 розділу 6 підручника у вигляді окремої функції, що повертає кількість доданків ряду.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця 6.1 Завдання на обчислення сум нескінченних рядів | | | |
| № | Функція | Ряд | Рекурентна формула |
| 1 | 2 | 3 | 4 |
| 0 | e |  |  |
| 1 |  |  | , для х<1 |
| 2 |  |  | , для х<1 |
| 3 |  |  | , для х<1 |
|  | | | |
| Продовження таблиці 6.2 | | | |
| 1 | 2 | 3 | 4 |
| 4 |  |  | , для х>1 |
|  |  |  |  |
| 5 | cos(x) |  | , для х<π |
| 6 | sh(x) |  | для х<1 |
| 7 | arctg(x) |  | для х<1 |
| 8 | ln(1+x) |  | , для х<1 |
| 9 |  |  | , для х<π |

Вимоги до звіту

Завантажити на сайт дистанційного навчання код проекту та скриншоти результатів тестування проекту. У функції main має бути перевірка вхідних даних на коректність (якщо є обмеження у завданні),  та виклики створених функцій.

## Контрольні питання

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

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

# Обробка данных за допомогою циклу for

Мета роботи:

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

Короткі теоретичні відомостіІ

### Оператор циклу for

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

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



Рисунок 7.1– Синтаксис оператора **for**

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

Оператор циклу **for** виконується наступним чином:

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

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

Як приклад використання циклу **for** розглянемо функцію обчислення числа Фібоначчі із заданим номером. Це послідовність чисел у якій нульове число дорівнює 0 і перше дорівнює 1. Решта чисел обчислюється як сума двох попередніх. Тобто послідовність має такий вигляд 0 1 1 2 3 5 8 13 21 34 …

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

uint fibo(uint n){

if(n==0 || n==1)

return n;

uint f; // Число Фібоначчі

uint f0=0, f1=1; //Початкові значення двох попередніх чисел

for(uint i = 2; i<=n; i++){

f=f0+f1; //Обчислюємо число Фібоначчі

// Змінюємо значення двох попередніх чисел

f0=f1;

f1=f;

}

return f;

}

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

unsigned long long fact(uint n) {

unsigned long long f=1;

for( uint i = 1; i<=n; i++ )

f\*=i;

return f;

}

### Особливості використання циклу for

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

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

unsigned long long fact(uint n) {  
 unsigned long long f=1;  
 uint i = 1;  
 for( ; ; ){  
 f\*=i;  
 if(i>=n)

return f;  
 i++;  
 }  
}

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

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

cout<<»/nВведiть цiле число\n «;  
 int x;  
 cin>>x;  
 int sum=0;  
 for( ; x!=0; ){  
 sum+=x%10;  
 x/=10;  
 }  
 cout<<»/nСума цифр у числi дорiвнює «<<sum<<»\n «;

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

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

Ось як, наприклад може виглядати функція обчислення факторіалу:

unsigned long long fact1(uint n) {

unsigned long long f=1;

for( uint i = 1; i<=n; f\*=i, i++ );

return f;

}

А так може виглядати функція обчислення числа Фібоначчі:

int fibo1(unsigned int n){

if(n==0 || n==1) return n;

unsigned int f=0;

for(int i=2,f0=0,f1=1; i<=n; f=f1+f0, f0=f1, f1=f, i++);

return f;

}

### Табулювання функцій

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

У якості прикладу розглянемо наведену нижче, на рисунку 7.2, програму табулювання функції y = a\*e (-b\*x) \*sin(w\*x). У програмі використано рекомендації даного пункту.

Результати роботи програми наведено на рисунку 7.3.

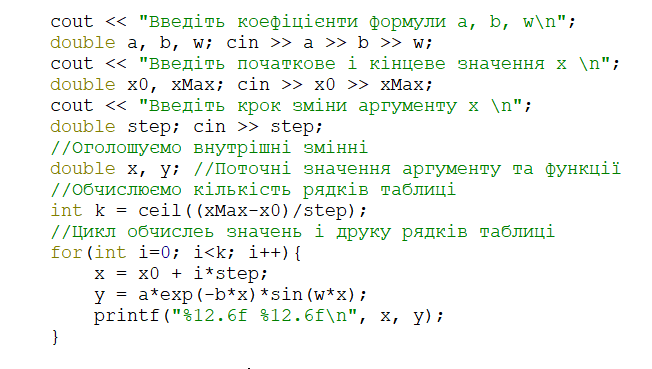


Рисунок 7.2 – Програма виведення таблиці значень функції

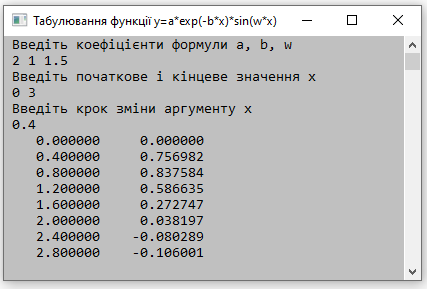


Рисунок 7.3 – Результати табулювання значень функції

### Обробка послідовностей цілих чисел

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

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

Розробка допоміжної функції

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

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

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

Тепер можна перейти до реалізації функції. Задачу будемо вирішувати шляхом аналізу усіх комбінацій чисел х1 та х2. Перше число х1 будемо змінювати від 1 до х-1. А для кожного значення х1 будемо аналізувати значення х2 від х1 до х2-1. Перебір значень х2 починаємо від поточного значення х1 для того, щоб уникнути аналізу дзеркальних комбінацій, наприклад 3 і 5 та 5 і 3.

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

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

Розробка функції для вирішення головної задачі

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

void intChain(){   
 cout<<»Введiть число n1>1 та число n2>n1 «;  
 uint x1, x2, a, b;  
 cin>>x1>>x2;  
 for(int i=x1; i<=x2; i++){  
 if(hasSqr(i, a, b)){  
 printf(«%4d^2 = %2d^2 + %2d^2 \n», x, x1, x2);

}  
 }  
}

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

Результат виконання цієї програми маємо на рисунку 7.4.

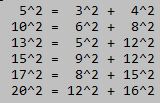


Рисунок 7.4 – Результат пошуку пар чисел, сума квадратів яких дорівнює квадрату заданого числу

### Випадкові числа

Для формування випадкових чисел в Qt використовується функція qrand(), яка повертає псевдо випадкове число з діапазону 0.. RAND\_MAX-1. RAND\_MAX=0х7FFF. 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 qsrand() використовують для завдання початкового числа (зерна) псевдо випадкової послідовності. В якості аргументу у цю функцію слід передати ціле число. Якщо цю функцію не використовувати, то функція qrand() буде завжди формувати ту саму послідовність.

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

Виклик функції qsrand() може виглядати так:

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

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

int x = qrand() % N;

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

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

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

float x = qrand() / (float) RAND\_MAX;

### Обробка послідовності випадкових чисел

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

Для отримання випадкової послідовності будемо використовувати функцію qrand() із урахуванням рекомендацій пункту 7.1.5.

Текст програми наведено на рисунку 7.5.

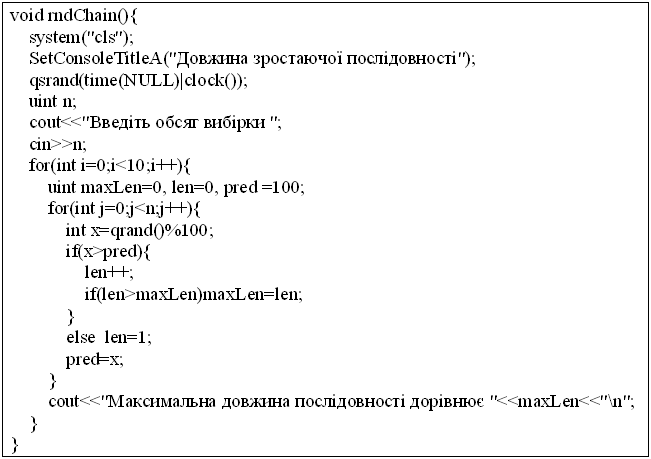


Рисунок 7.5 – Функція для визначення максимальної довжини зростаючої послідовності у послідовності випадкових чисел

У програмі використовується два цикли for.

Перший забезпечує десяти- разове повторення обробки послідовності випадкових чисел.

Другий, внутрішній цикл, забезпечує формування послідовності заданого розміру і пошук найбільшої довжини зростаючої послідовності.

Результати виконання програми наведено на рисунку 7.6.

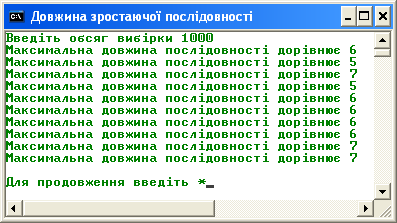


Рисунок 7.6 – Результати визначення максимальної довжини зростаючої послідовності у послідовності випадкових чисел

Завдання на лабораторну роботу

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

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

Друга програма має забезпечити обробку цілих чисел у відповідності з завданням з таблиці 7.1, обраним відповідно до останньої цифри номера залікової книжки.

|  |  |
| --- | --- |
| Таблиця 7.1 – Варіанти завдань з обробки цілих чисел | |
| № | Завдання |
| 1 | 2 |
| 0 | Написати програму, що буде виводити на консоль у зростаючому порядку і підраховувати тризначні цілі числа, в яких немає однакових цифр. Слід попередньо написати функцію, що визначає, чи є однакові цифри в числі. |
| 1 | Написати програму, що буде виводити на консоль у зворотному порядку і підраховувати тризначні цілі числа, в яких є поспіль дві однакові цифри. Попередньо написати функцію, що визначає, чи є в числі поспіль дві однакові цифри. |
| 2 | Написати програму, що буде виводити на консоль тризначні цілі числа, сума цифр яких дорівнює введеному числу N (1<N<27). Визначити кількість цих чисел. Попередньо написати функцію, яка обчислює суму цифр числа. |
|  | |
| Продовження таблиці 7.1 | |
| 1 | 2 |
| 3 | Написати програму, що буде виводити на консоль всі прості числа, які менші за N, і підраховувати їх кількість. Попередньо написати функцію тестування числа на простоту. Простим називають ціле число, яке без остачі ділиться тільки на 1 і на себе. Ознакою є наявність залишку від ділення числа X на числа від 2 до X / 2. |
| 4 | Написати програму, що буде виводити на консоль цілі числа з інтервалу від n1 до n2 і суми дільників (за винятком 1 і самого числа) для кожного з них. Для знаходження дільників слід перевіряти результат ділення числа N на числа від 2 до N / 2. Попередньо написати функцію підрахунку суми дільників одного числа. |
| 5 | Написати програму, що буде виводити на консоль цілі числа з інтервалу від n1 до n2 до складу яких входить задана цифра. . Попередньо написати функцію, що визначає, чи є в числі потрібна цифра. |
| 6 | Написати програму, що буде знаходити мінімальне значення функції f(x)=10x2+215x +100 на інтервалі цілих чисел від n1 до n2. Попередньо написати функцію, що обчислює значення f(x). |
| 7 | Написати програму, що буде виводити на консоль цілі числа з інтервалу від n1 до n2 і перелік їх простих дільників (за винятком 1 і самого числа). Для знаходження дільників слід перевіряти результат ділення числа N на числа від 2 до N / 2. Попередньо написати функцію пошуку дільників одного числа. |
| 8 | Написати програму, що буде виводити на консоль значення факторіалу цілих чисел з інтервалу від n1 до n2. Попередньо написати функцію, що обчислює факторіал числа. |
| 9 | Написати програму, що буде виводити на консоль значення числа Фібоначчі з номерами від n1 до n2. Попередньо написати функцію, що обчислює число Фібоначчі із заданим номером. |

|  |  |
| --- | --- |
| Таблица 7.2 – Варіанти завдань з обробки послідовностей випадкових чисел | |
| Варіант | Характеристика послідовності, що має бути обчислена |
| 0 | Максимальне та мінімальне число |
| 1 | Найбільша довжина послідовності, числа якої зменшуються |
| 2 | Середнє значення |
| 3 | Середнє значення квадратів чисел |
| 4 | Кількість повторень заданого числа |
| 5 | Кількість парних чисел |
| 6 | Кількість непарних чисел |
| 7 | Різниця між максимальним та мінімальним числом (розмах) |
| 8 | Кількість чисел кратних заданому числу |
| 9 | Номери заданого числа у послідовності |

Вимоги до звіту

Завантажити на сайт дистанційного навчання код проекту та скриншот результатів тестування проекту.

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.

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

Основи роботи із вказівниками

Мета роботи:

* Ознайомитися з поняттям вказівник.
* Навчитися оголошувати та ініціалізувати вказівники.

Короткі теоретичні відомості

Вказівник або покажчик – це особливий тип даних, значенням якого є адреса певного байту оперативної пам’яті. Цей тип даних найчастіше використовується у системному програмуванні, але і прикладні програмісти успішно використовують цей інструмент. Слід сказати, що не усі мови програмування надають можливість використовувати 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 простий у використанні засіб програмування і часто стають джерелом помилок, виявити які у програмі дуже складно. Тому у деяких мовах прикладного програмування (наприклад, Java) принципово відмовилися від вказівників.

### Оголошення та ініціалізація вказівників

Оголошення вказівника має вигляд, представлений на рисунку 8.1.



Рисунок 8.1 – Синтаксис оголошення вказівника

У цьому оголошенні тип даних визначає тип елементу 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, 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 ;

### Звернення до даних через вказівники

Якщо відома адреса якихось даних, то мабуть можна отримати і значення цих даних. Операцію отримання даних через їх адресу називають операцією розадресації і позначають знову ж таки символом \*, що ставиться перед ім’ям 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 вашої програми.

### 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 зміювати 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;

### 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.

Операції із шістнадцятковими числами

Адреси пам’яті найчастіше представляються у шістнадцятковому форматі. Тому програміст має вміти виконувати операції додавання і віднімання чисел у такому форматі.

Наприклад, якщо початкова адреса масиву чисел типу int дорівнює 0xf45bc і ми хочемо визначити адресу елементу з індексом 9, то спочатку треба визначити реальне зміщення елементу від початку масиву, враховуючи довжину одного елементу в байтах. Довжина елементу типу int дорівнює 4. Тож зміщення буде дорівнювати 4\*9 = 36. Далі це число слід перевести у 16-ву систему числення. Число 36 можна представити як 2\*16 + 4\*1, тобто у 16-вій системі воно буде виглядати як 0x24. Після цього можна додати це зміщення до початкової адреси. Додавання доцільно робити у стовпчик.

У молодшому розряді маємо с + 4, що у десятковій системі буде 12 + 4 = 16, або у 16-вій = 0х10. Таким чином останній розряд результату буде 0 і одиничка переносу.

У наступному розряді маємо b + 2 + 1 переносу, що у десятковій системі виглядає як 11 + 2 + 1 = 14 = 0хe. Маємо передостанній розряд результату, що дорівнює е.

Інші розряди не змінилися. Відповідь – 0xf45e0.

### Нетипізовані вказівники

У мові С можна використовувати вказівники, які не пов’язані з якимось конкретним типом і сумісні вказівниками на будь які типи даних. Використання безтипових вказівників дозволяє підвищувати ефективність створюваних 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, який дозволяє отримати доступ до двох половин числа типу int (int удвічі більший за short).

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.

### 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, так само, як ім’я масиву є вказівником на ділянку пам’яті, де знаходиться цей масив.

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

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:

#define double TypeFunc(double, double);

Оголошення типу, як бачимо, схоже на оголошення 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 на нашу думку ці способи важче сприймаються і запам’ятовуються. Докладно познайомитися із способами оголошень вказівників на функції можна у підручнику[1], сторінка 212.

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.

З огляду на той факт, наші функції одного типу, ми можемо написати одну узагальнену програму роздруківки таблиці значень для будь якої функції визначеного типу. Посилання на потрібну функцію будемо передавати до цієї універсальної програми через параметр.

Узагальнена програма (теж функція) може виглядати так:

void printFunс(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

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:

int main()

{

printFunc(0,1,0.1, 2.5, 5, calcY);

printFunc(1,2,0.2, 3, 7, calcZ);

return 0;

}

Завдання для САМОСТІЙНОЇ РОБОТИ

* Опрацювати лекцію по темі та розділ 7 підручника і дати відповіді на питання 1, 5, 6, 10 у вигляді текстового документа.
* виконати без комп’ютера, а потім перевірити результат за допомогою комп’ютера (дивись приклад із підручника [4], стр.120). До звіту не включати, але схоже завдання буде на тесті.

//Дано:

int mvr={99, 11, 22, 33, 44, 55};

int \*pm = mvr;

int a, \*px = pm;

//Треба знайти результат кожного з наступних операторів:

pm++;

a = \*pm++;

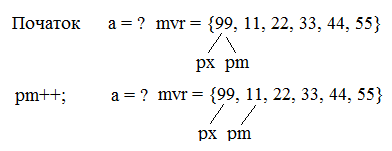
a = (\*pm)++;

a = \*++pm;

a = ++\*pm;

\*px++ = \*pm++;

Для зручності роботи результати виконання кожного шагу рекомендуємо представляти у звіті за такою схемою:



* виконати без комп’ютера, а потім перевірити результат за допомогою комп’ютера приклад 3 із підручника [4], стр.124. До звіту не включати, але схоже завдання буде на тесті.

//Яким буде х після виконання даної програми:

int x, \*vk;

vk=&x;

x=5;

\*vk = x \* 5;

x+=\*vk;

* виконати без комп’ютера, а потім перевірити результат за допомогою комп’ютера приклад 8 із підручника [4], стр.124. До звіту не включати, але схоже завдання буде на тесті.

//Яким буде n після виконання даної програми:

double \*px;

double \* const py = (double\*)200;

int n = 0;

px = py +50;

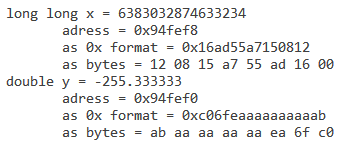
px += 10;

px = px >= (double\*) 600 ? px - 10 : NULL ;

if (px !=NULL)

n = px - py;

* написати функцію виведення на консоль значень чисел типів int, long long, float, double, char у десятковому та шістнадцятковому (якщо можливо) форматі, їх адрес у оперативній пам’яті та представлення у вигляді окремих байтів. Для long long та double результат може виглядати так:



Для виведення у 16-му форматі використовуйте функцію printf() із специфікацією формату %#lx. Для побайтового представлення використовуйте приклад з роділу 8.1.5 методички та приклад на сторінці 123 підручника[1].

* Визначити тип для функції що використовувалася у попередній та третій роботі. Написати функцію табулюванням функцій такого типу. За допомогою цієї функції отримати таблицю значень, передавши туди як параметр ім’я функції, що використовувалася у попередній та третій роботі.

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* Короткий опис основних понять, пов’язаних з вказівниками.
* Тексти виконаних завдань з проміжними та конечними результатами.
* Результати тестування проекту у вигляді копій консолі.
* Висновки.

Контрольні питання

* Визначення поняття вказівник.
* Оголошення та ініціалізація вказівників.
* Звернення до даних через вказівники.
* Використання кваліфікатора const для вказівників.
* Адресна арифметика.
* Нетипізовані вказівники.
* Протестувати програму обробки вказівників наданої викладачем.

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

Рядки символів

Мета роботи:

* Ознайомитися з поняттям рядок символів.
* Ознайомитися з деякими алгоритмами обробки рядків символів.
* Створити програму, яка обробляє рядок символів відповідно до індивідуального завдання.

Короткі теоретичні відомості

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

Коли у тексті програми на мові С з’являлася константа, охоплена подвійними лапками, наприклад, “Невже це не рядок символів?”, то вона розглядалася, як масив 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’.

### Оголошення рядка

Символьні рядки, які є змінними програми, оголошуються, як масиви типу char, рисунок 9.1.



Рисунок 9.1 – 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 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!”;

### 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[10];

cin.getline(str,10);

Першим параметром цього методу є посилання на рядок символів, а другим – максимальна кількість символів, що буде прийнята з консолі.

Рядок можна зчитати і за допомогою функції gets(), але ця функція не контролює кількість введених символів, що може призвести до непередбачуваних результатів у разі введення більшої кількості символів, ніж виділено пам’яті. Окрім того, використання функції gets() потребує підключення заголовного файлу <cstdio> або < stdio.h>.

### 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 у таблиці.
* перша половина цієї таблиці (символи з номерами з 0 по 127) стандартна. Вона містить цифри, латинські літери та інші необхідні 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 країн та різних кодових таблиць,  встановлених у операційній системі.

Нижче наведений варіант цієї таблиці, скопійований із Internet:

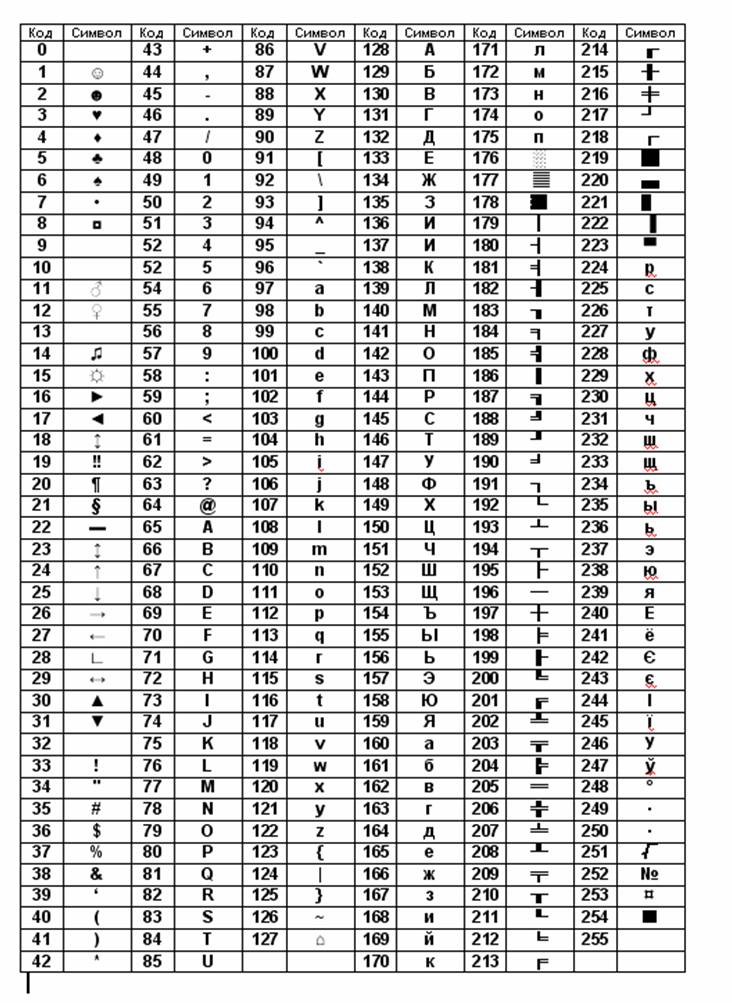


Рисунок 9.2 – Таблиця кодів символів

### 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] = ‘а’ перетворить рядок у ”Hаllo!”.

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.

Функція, що обчислює кількість символів у рядку

У цій функції використовується та особливість рядка, що він закінчується символом ‘\0’.

int length(char \*str){

int count = 0;

while(str[count] != '\0')

count ++;

return count;

}

Функція копіювання частини рядка

До функції передається посилання str на рядок, з якого будуть копіюватися символи, та посилання copy на ділянку пам’яті, куди будуть копіюватися потрібні символи. Передається також позиція index, з якої треба починати копіювання, та число count , що визначає, скільки символів потрібно скопіювати ,

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’;

}

У наведеній функції спочатку перевіряється коректність переданих параметрів і, в разі потреби, вони корегуються. Після цього реалізується послідовне копіювання символів.

Функція знаходження позиції групи символів у рядку

До функції передається посилання str на рядок, у якому буде йти пошук позиції, та посилання sub на групу символів, позиція якої нас цікавить.

Для визначення довжини рядків використовуємо функцію length, яка була наведена вище у цьому підрозділі.

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.

Функція, що вилучає із рядка зайві пробіли

Функція залишає лише один пробіл там, де їх декілька поспіль і повертає кількість вилучених пробілів.

Вилучення пробілів реалізується шляхом зсуву символів правої частини рядка ліворуч на місце зайвих пробілів і перенесення символу кінця рядка.

int delExtraBlank(char \*str){

int count = 0;

for(int i = 0; str[i] != '\0'; i++){

if(str[i] == ' ' && str[i+1] == ' ' ){

//seeking for position without blank

//and accumulate number of extra blank

int p = ++i;

while(str[p] == ' '){

count++; p++;

}

// shift left simbols of string

int j = p;

for( ; str[j] != '\0'; j++)

str[i + j - p] = str[j];

str[i+j-p] = '\0';

}

}

return count;

}

Перетворення цілого числа у рядок символів

Комп’ютер оперує з числами, як даними числових типів. Але для користувачів результатів обчислень мають бути представлені у вигляді рядків символів. Так само і зворотний обмін числами. Користувач вводить числа як рядок символів, який потрібно привести до числового типу.

Проблема таких перетворень полегшується завдяки тому, що числовий код десяткової цифри менше від коду відповідного символу на 48.

Нижче наведено функцію перетворення цілого числа у рядок символів.

void intToStr(int x, char str[]){

int i=0, z=abs(x); str[0]='\0';

do{

//Звільняємо місце для цифри

for(int j=i++; j>=0; j--) str[j+1]=str[j];

//Наступна цифра

str[0]=z%10+48;

}while(z/=10);

if(x<0){

for(int j=i; j>=0; j--) str[j+1]=str[j];

str[0]='-';

}

}

Цифри числа виокремлюються за допомогою операції %10, починаючи з останньої. Після цього число зменшується у 10 разів, внаслідок чого передостання цифра стає останньою.

Відокремлена цифра перетворюється у код відповідного символу шляхом додавання числа 48.

Місце в рядку для нової цифри надається на початку рядка шляхом зсуву праворуч попередньо доданих цифр. Спочатку у рядку записується тільки символ кінця рядка.

Якщо число було від’ємне, то до початку рядка додається символ мінус.

Перетворення рядка символів у ціле число

Ця функція обробляє символи числа до появи першого нецифрового символу.

Код символу перетворюється у код числа шляхом зменшення на 48.

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 х 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.

### 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. Прототипи цих функцій знаходяться у заголовному файлі <ctype.h>.

Перелік найбільш популярних з цих функцій наведено у таблиці 9.1 підручника[1], сторінка 151. Там можна знайти більше інформації про ці функції і познайомитися з прикладами їх використання.

На рисунку 9.3 наведено фрагмент цієї таблиці 9.1 підручника. Ваше завдання протестувати наведені функції.

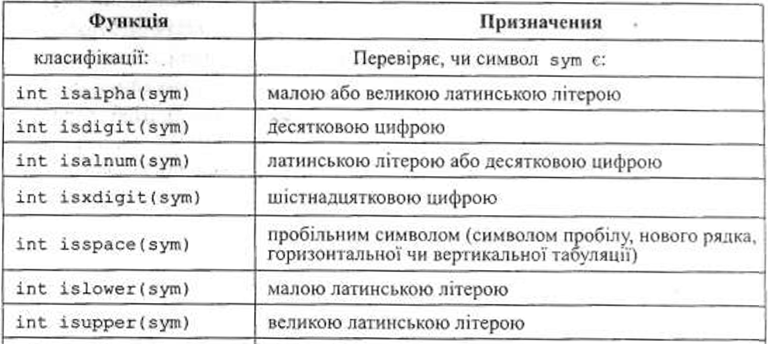


Рисунок 9.3 – Функції тестування символів

Друга група функцій призначена 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.

Третя група функцій призначена 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[1], сторінка 155. Там можна знайти інформацію про ці функції і познайомитися з прикладами їх використання.

Завдання для роботи

В лабораторній роботі слід створити проект, де мають бути написані і протестовані в методі main() такі функції:

Функція, що тестує символи заданого рядка і виводить на консоль інформацію про кожен символ, використовуючи функції таблиці, зображеної на рисунку 9.3. Якщо символ рядка відноситься до декількох категорій, треба вивести кожен варіант.

Функція, що повертає найбільше число із рядка символів, у якому числа розділені пробілами. Для вирішення задачі використовувати функції strtok() та atoi() і приклад, наведений на сторінці 154 підручника.

Функція відповідно до вимог варіанту з таблиці 9.1 без використання стандартних функцій для обробки рядків символів.

Номер варіанту вибирайте відповідно до номера залікової книжки.

|  |  |
| --- | --- |
| Таблиця 9.1 – Завдання на обробку рядків символів | |
| **№** | **Завдання** |
| 0 | Отримати задану кількість символів, від початку рядка |
| 1 | Отримати задану кількість символів, від кінця рядка |
| 2 | Перетворити рядок шістнадцяткових цифрових символів у число |
| 3 | Перетворити ціле число без знаку у шістнадцятковий рядок символів |
| 4 | Вилучити початкові пробіли з рядка символів |
| 5 | Вилучити хвостові пробіли з рядка символів |
| 6 | Знайти у рядку послідовність цифрових символів і повернути її, |
| 7 | Видалити задану кількість символів, починаючи від заданої позиції |
| 8 | Вставити задану послідовність символів, починаючи від заданої позиції |
| 9 | Отримати першу позицію заданої послідовності символів з кінця рядка |

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* Тексти функцій для індивідуальних завдань з коментарями.
* Результати тестування проекту у вигляді копій консолі.
* Висновки.

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.

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

Одновимірні масиви

Мета роботи:

* Ознайомитися з поняттями масив.
* Навчитися оголошувати та ініціалізувати одновимірні масиви.
* Навчитися оперувати масивами, як параметрами функцій.
* Створити програму для обробки масивів.

Короткі теоретичні відомості про масиви

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

* масив складається з елементів, які мають однаковий тип;
* елементи масиву послідовно розташовані в одній ділянці оперативної пам’яті без проміжків між елементами;
* кожен з елементів масиву має свій порядковий номер, що зветься індексом;
* нумерація елементів починається з 0;
* до елементів масиву можна звертатися використовуючи ім’я масиву і індекс;.
* масив може бути одновимірним, або багатовимірним, тобто таким у якого кожний елемент є також масивом;
* у мові С, С++ ім’я масиву зберігає адресу першого елементу цього масиву, тобто є вказівником на його початок.

### Оголошення одновимірного масиву та звернення до його елементів

Оголошення одновимірного масиву має вигляд, представлений на рисунку 10.1.



Рисунок 10.1 – Синтаксис оголошення одновимірного масиву

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

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

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

Приклад оголошення масиву з ім’ям 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 i 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 елементу.

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

Працюючи з масивами у програмах на мові С слід пам’ятати, що ніякого контролю за значеннями індексів, що використовуються для доступу до елементів масиву, нема. Можна звернутися до «елементу масиву» з будь яким номером, але отримати невідомо що. Ще гірше може бути, якщо змінити значення цього «елементу масиву». Наслідки можуть бути катастрофічними для програми.

### Приклад використання одновимірного масиву

Нижче наведено приклад створення масиву для збереження чисел 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];

}

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

Крім того, слід брати до уваги той факт, що масив «не знає», скільки у нього елементів, тому передаючи масив до функції слід передавати і кількість елементів масиву, що має бути 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, але масив можна повернути через параметри функції. Ось, наприклад, як може виглядати функція, що повертає масив чисел Фібоніччі.

void fibo(int a[], int size){

a[0] = 0;

a[1] = 1;

for (int i = 2 ; i < size; i++)

a[i] = a[i – 2] + a[i – 1];

}

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

Щоправда, слід мати на увазі, що функція може повертати 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;

}

Функції обробки масивів чисел

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

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

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

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

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

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

### Функція формування випадкового масиву

Ця функція дещо відрізняється від функції генерації послідовності випадкових чисел, що була розглянута раніше. Різниця полягає у тому, що сформовані числа записуються у елементи масиву. Текст функції наведено на рисунку 10.3.

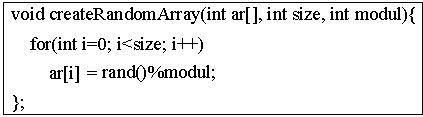


Рисунок 10.3 – Функція формування випадкового масиву

### Функція виведення масиву на консоль

Ця функція досить проста і не потребує коментарів. Ця функція досить проста і не потребує коментарів. Текст функції наведено на рисунку 10.4.

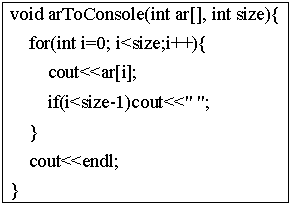


Рисунок 10.4 – Функція виведення масиву на консоль

### Функція введення масиву з консолі по елементам

Це найпростіший варіант, який полягає у послідовному введенні розміру та кожного елементу масиву з консолі як чисел за допомогою об’єкту cin.

Розглянемо цей варіант.

Текст функції наведено на рисунку 10.5.

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

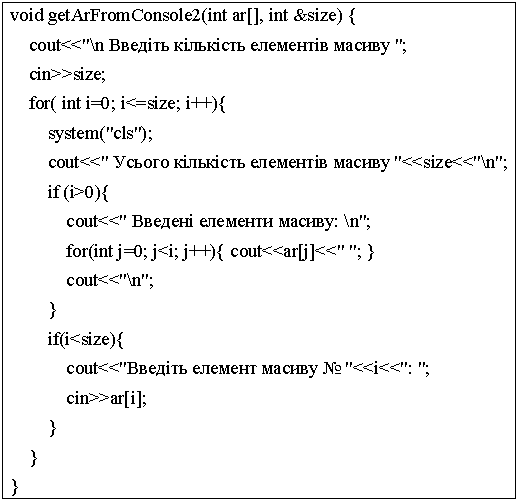
****

Рисунок 10.5 – Функція введення масиву по елементам

Функція могла б виглядати так само просто як і функція виведення масиву, що наведена у пункті 10.2.2, і ви можете її саме так і написати, але у наведеній нижче реалізації додано операції очищення екрану і виведення елементів, що були вже введені, і очищення екрану для того, щоб кожен елемент масиву вводився на тому ж самому місці, що й попередній.

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

### Функція перетворення рядка символів у числовий масив

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

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

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

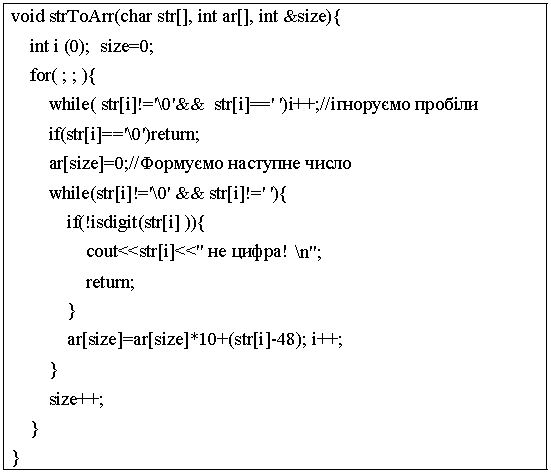


Рисунок 9.2 – Функція перетворення рядка символів у масив

Цю функцію можна використовувати для введення числового масиву з консолі, записуючи числа у одному рядку через пробіл. Такий рядок можна зчитати, а далі перетворити у послідовність чисел.

### Функція перевороту масиву

Текст функції наведено на рисунку 10.6.

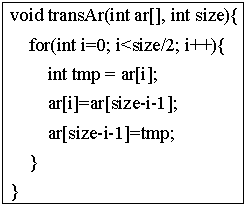


Рисунок 10.6 – Функція для перевороту масиву

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

### Функція вилучення елементу з масиву

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

Для перебору елементів масиву у функції використовується цикл **while,** тому що індекс елементу у разі зсуву елементів ліворуч не потрібно змінювати. Текст функції наведемо на рисунку 10.7.

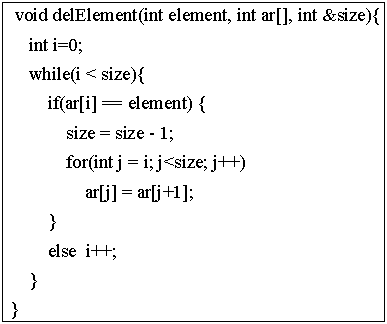


Рисунок 10.7 – Функція вилучення елементу з масиву

### Функція формування масиву накопичених значень

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

Текст функції наведено на рисунку 10.8.

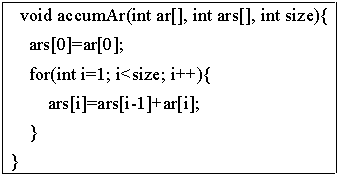


Рисунок 10.8 – Функція для формування масиву накопичених значень

Завдання для САМОСТІЙНОЇ РОБОТИ

В лабораторній роботі слід створити проект, відповідно до вимог варіантів з таблиць 10.1. Номер варіанту вибирайте відповідно до останньої цифри номера залікової книжки.

Таблиця 10.1 передбачає реалізацію таких функцій у проекті:

* створення масиву заданим способом з використанням вказівників;
* визначення якоїсь числової характеристики для створеного масиву з використанням індексів;
* формування нового масиву на основі створеного. Новий масив створити у динамічній пам’яті, а функція має повертати вказівник на цей масив.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця 10.1 – Завдання на роботу з масивами | | | |
| № | Створення масиву | Числові характеристики | Формування нового масиву |
| 1 | 2 | 3 | 4 |
| 0 | Random | Розмах елементів  (max-min) | Видалити парні елементи із масиву |
| 1 | По елементам | Різниця між сумами елементів у парних та непарних позиціях | Оборот масиву |
| 2 | Random | Кількість елементів, що перевищують середнє геометричне | Вставка елемента в задану позицію |
| 3 | По елементам | Різниця між середніми арифметичним та геометричним | Видалення заданого елементу із масиву |
| 4 | Random | Різниця між сумами парних та непарних елементів | Задане число циклічних зсувів ліворуч |
| 5 | По елементам | Середнє відхилення елементів масиву від середнього арифм. | Вставка суми елементів у початок масиву |
| 6 | Random | Середні арифметичні значення парних та непарних елементів | Задане число циклічних зсувів праворуч |
| 7 | По елементам | Середнє значення модуля різниці між сусідніми елементами масиву | Вставка середнього арифметичного значення в середину масиву |
| 8 | Random | Кількість елементів, що перевищують середнє значення | Переформатувати масив, спочатку непарні, потім парні |
| 9 | По елементам | Сума елементів, що перевищують середнє значення | Додати мінімальне значення у початок масиву, а максимальне к кінець |

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* Тексти функцій для індивідуальних завдань з коментарями.
* Результати тестування проекту у вигляді копій консолі.
* Висновки.

Контрольні питання

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

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

Сортування масивів

Мета роботи:

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

Методи сортування масивів

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

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

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

ar[0] <= ar [1] <= ar [2] <= ar [3] <= …….<= ar [n]

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

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

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

### Сортування вибором

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

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

Аналіз описаних вище дій показує, що для програмної реалізації цього методу сортування буде потрібно два цикли for.

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

Внутрішній цикл повинен забезпечити послідовне порівняння елементу, зафіксованого першим циклом, з усіма елементами, які слідують в масиві за ним.  
У тілі внутрішнього циклу відбувається порівняння елементів, індекси яких задаються параметрами зовнішнього і внутрішнього циклу. Якщо в результаті порівняння знаходиться елемент, що менший ніж зафіксований, то порівнювані елементи міняються місцями.

Схема алгоритму сортування масиву методом вибору показана на рисунку 11.1.

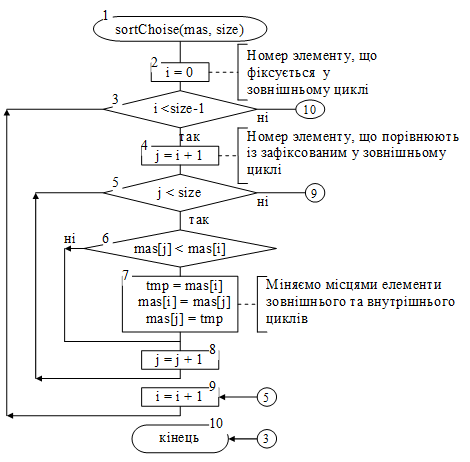


Рисунок 11.1 - Алгоритм сортування елементів масиву у зростаючому порядку за методом вибору

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

Приклад сортування елементів масиву у зростаючому порядку за методом вибору

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

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

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

Проілюструємо графічно алгоритм цього методу сортування. Початкове розташування елементів масиву показано на рисунку 11.2.



Рисунок 11.2 – Початкова схема масиву, що підлягає сортуванню

Після обміну значеннями елементів A[0] и A[1] масив стане таким, як показано на рисунку 11.3.



Рисунок 11.3 – Схема масиву після першого обміну значеннями

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

Аналіз елементу A[3] показує, що він менший за A[0], тому знову потрібен обмін значеннями, але вже між A[3] і A[0]. Результат обміну показано на рисунку 11.4.



Рисунок 11.4 – Схема масиву після другого обміну значеннями

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

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

Тепер необхідно здійснити другий перегляд масиву, але в якості змінюваного береться вже другий елемент масиву. З другим елементом будуть порівнюватися елементи, починаючи з третього. В результаті чого на друге місце буде поставлений мінімальний елемент серед елементів з 2-го по 5-й. Результати другого проходу по масиву представлені на рисунку 11.5.



Рисунок 11.5 – Схема другого проходу при сортуванні масиву методом вибору

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

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



Рисунок 11.6 – Схема останніх проходів по масиву при сортуванні методом вибору

Функція сортування масиву методом вибору

Функція, представлена на рисунку 11.7 реалізує за допомогою циклів for алгоритм, зображений на рисунку 11.1.

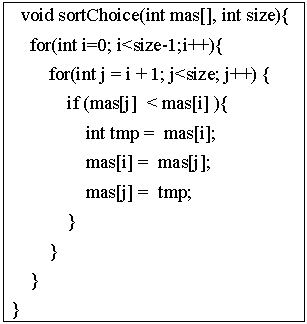


Рисунок 11.7 – Функція сортування елементів масиву у зростаючому порядку за методом вибору

Наведена функція не єдиний варіант реалізації методу вибору.

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

### Сортування обміном (метод бульбашки)

Алгоритм заснований на принципі порівняння та обміну значеннями (у разі необхідності) між сусідніми елементами. Кожен елемент масиву, починаючи з першого, порівнюється з наступним, і якщо він більше наступного, то елементи міняються місцями. В результаті після першого проходу, при сортуванні на зростання, найбільший елемент виявиться на останньому місці.

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

Схему алгоритму сортування елементів масиву у зростаючому порядку за методом обміну показана на рисунку 11.9.

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

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

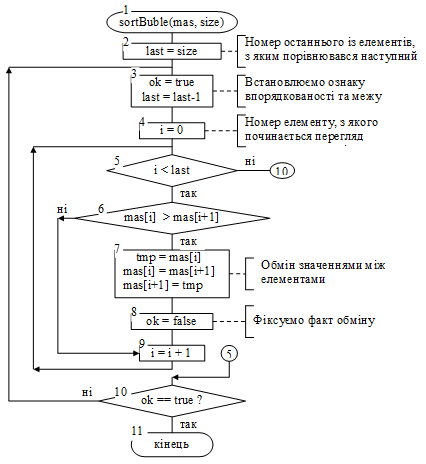


Рисунок 11.8 - Алгоритм сортування елементів масиву у зростаючому порядку за методом обміну

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

У алгоритмі, що зображений на рисунку 11.8, сортування реалізується за допомогою двох циклів. Зовнішній цикл виконується поки по завершенні чергового проходу по масиву не виявиться, що перестановок елементів не було. У цьому випадку змінна ok збереже значення true, і це означатиме, що масив відсортований. Наявність обмінів між елементами при проході по масиву можна перевірити тільки після завершення проходу, тому зрозуміло, що в даному випадку доречним буде цикл do ... while.

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

Приклад сортування елементів масиву у зростаючому порядку за методом обміну

На рисунках 11.9 – 11.13 детально показані зміни масиву в процесі сортування обміном під час першого проходу.



Рисунок 11.9 – Масив до сортування



Рисунок 11.10 – Масив післе першого обміну елементів



Рисунок 11.11 – Масив після другого обміну елементів



Рисунок 11.12 – Масив після третього обміну елементів



Рисунок 11.13 – Масив після першого проходу

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

Другий прохід по масиву розглянемо менш детально. Його результати представлені на рисунку 11.14.



Рисунок 11.14 – Другий прохід по масиву

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

Результати останнього проходу по масиву показані на рисунку 11.15.



Рисунок 11.15 – Результати третього проходу по масиву

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

Функція сортування масиву методом обміну

Функція, представлена на рисунку 11.16 реалізує за допомогою циклів for алгоритм, зображений на рисунку 11.1.

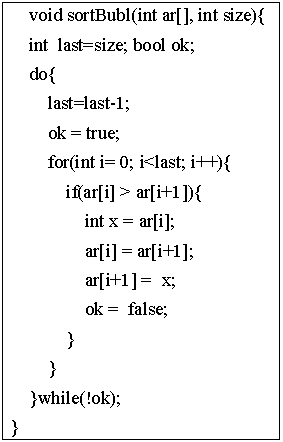


Рисунок 11.16 - Функція сортування елементів масиву у зростаючому порядку за методом обміну

Роботу функції, представленої на рисунку 11.16 можна прискорити, якщо після обміну елементів фіксувати номер елементу, що останнім змінювався (значення змінної і). Це значення можна використовувати і як ознаку завершення сортування, так і в якості максимального значення параметру внутрішнього циклу. Це удосконалення алгоритму дозволяє дещо зменшити кількість циклів сортування. Саме такий спосіб слід реалізувати у проекті до лабораторної роботи.

### Сортування вставкою

Суть алгоритму полягає у наступному. На кожному кроці елементи масиву поділені на впорядковану частину А[1], А[2], ......, А[i-1], яка розташовується на початку масиву, та невпорядковану частину А [i], ....... , а [N], і перший елемент з невпорядкованої частини вставляється в упорядковану. На початку сортування впорядкована частина складається всього з одного, першого елемента, а всі інші елементи розташовуються в другій частині масиву. Схема цього алгоритму наведена на рисунку 11.17,

Послідовні кроки алгоритму сортування полягають у тому, що перший елемент з невпорядкованої частини порівнюється з останнім елементом впорядкованої послідовності. Якщо виявляється, що порядок розташування цих елементів не відповідає вимогам сортування, то елемент з невпорядкованої частини вилучається і переноситься в упорядковану частину. Місце для цього елементу звільняється шляхом зсуву упорядкованих елементів вправо на місце витягнутого елемента. Зсув упорядкованих елементів на одну позицію вправо триває, поки не буде знайдено місце для елемента, вилученого з невпорядкованої послідовності.

Як видно з рисунку 11.17, в алгоритмі є два цикли.

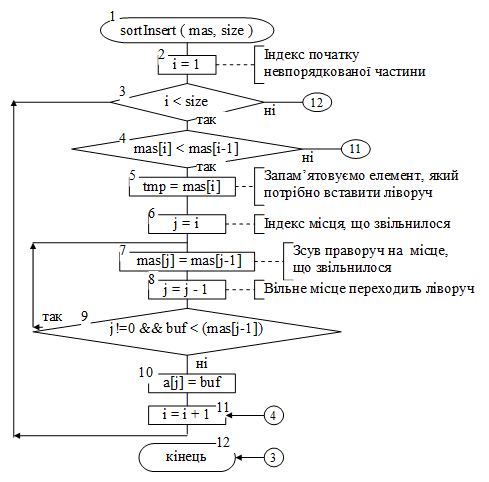


Рисунок 11.17 - Алгоритм сортування елементів масиву у зростаючому порядку за методом вставки

У зовнішньому циклі послідовно змінюється номер лівої межі невпорядкованою області від другого елементу (i=1) до кінця масиву. У тілі цього циклу порівнюються елементи, що знаходяться по обидва боки від межі, що розділяє впорядковану і невпорядковану частини. Якщо порядок порушений, то перший елемент невпорядкованою послідовності запам’ятовується у змінній tmp, внаслідок чого звільняється місце для зсувів упорядкованих елементів вправо.

Внутрішній цикл забезпечує послідовні зсуви упорядкованих елементів вправо, починаючи з останнього, доти, поки не буде знайдено місце для першого елемента з невпорядкованою області, що зберігався у змінній tmp.

Можливість вставки елементу визначається однією з двох умов.

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

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

Приклад сортування елементів масиву у зростаючому порядку за методом вставки

Результати послідовних кроків сортування представлені на рисунках 11.18 ‑ 11.22.



Рисунок 11.18 – Початкове розбиття масиву при сортуванні за методом вставки



Рисунок 11.19 – Результати виконання першого шагу сортування вставкою



Рисунок 11.20 – Результати виконання другого шагу сортування вставкою



Рисунок 11.21 – Результати виконання третього шагу сортування вставкою



Рисунок 11.22 – Результати виконання четвертого шагу сортування вставкою

Функція сортування масиву за методом вставки

Програмна реалізація алгоритму за методом вставки наведена на рисунку 11.23.

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

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

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

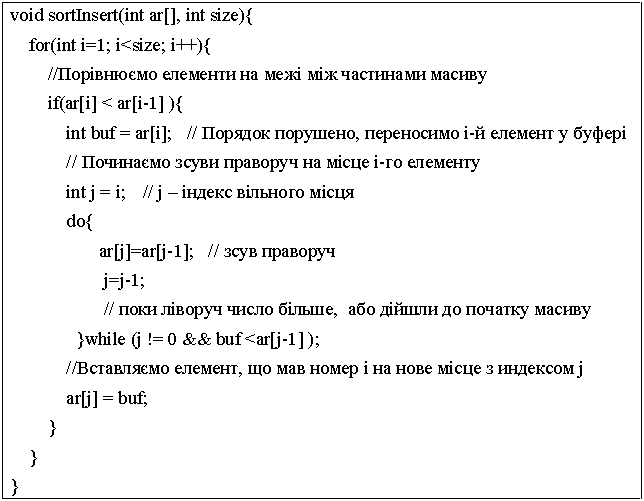


Рисунок 11.23 – Функція сортування елементів масиву у зростаючому порядку за методом вставки

Сортування за ускладненими правилами

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

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

Наприклад, нехай правило сортування масиву цілих чисел сформульовано таким чином: «Спочатку парні числа за спаданням, а потім непарні за зростанням».

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

bool goodDisposition(int x1, int x2){  
 if (x1 % 2 != x2 % 2) // Порівнюємо, чи однакові числа за парністю?  
 //Якщо одне число парне а друге непарне, то меншим буде парне  
 return x1 % 2 < x2 % 2;  
 else if( x1 % 2 == 0) // Якщо числа парні, має бути спадання  
 return x1>x2;  
 else // Якщо чила непарні, має бути зростання  
 return x1 < x2;  
}

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

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

Завдання для самостійної роботи

В лабораторній роботі слід створити програму, відповідно до вимог варіанту з таблиці 11.1. Номер варіанту вибирається відповідно до останньої цифри номеру залікової книжки.

Введення елементів масиву із рядка, де числа розділено пробілом. Слід реалізувати таку функцію, якщо раніше це не було зроблено.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця 11.1 – Завдання для самостійної роботи | | | |
| № | Метод сортування | Ускладнене правило сортування |
| 1 | 2 | 3 |
| 0 | Модифікований вибором | Спочатку додатні на збільшення, потім від’ємні на збільшення |
| 1 | Модифікований  обміном | Спочатку від’ємні на зменшення, потім додатні на зменшення |
| 2 | Вставкою | Спочатку парні, потім непарні, усі на зростання |
| 3 | Модифікований вибором | Спочатку не парні, потім парні, усі на спадання |
| Продовження таблиці 11.1 | | |
| 1 | 2 | 3 |
| 4 | Модифікований  обміном | Спочатку непарні за зростанням, потім парні числа за спаданням |
| 5 | Модифікований  обміном | Спочатку додатні на збільшення, потім від’ємні на збільшення |
| 6 | Модифікований обміном | Спочатку від’ємні на зменшення, потім додатні на зменшення |
| 7 | Вставкою | Спочатку парні, потім непарні, усі на зростання |
| 8 | Вставкою | Спочатку не парні, потім парні, усі на спадання |
| 9 | Модифікований вибором | Спочатку непарні за зростанням, потім парні числа за спаданням |

У програмі мають бути реалізовані такі функції:

* функція введення елементів масиву із рядка, де числа розділено пробілом;
* функція виведення масиву на консоль одним рядком через пробіл;
* функція сортування масиву заданим для варіанту методом з використанням вказівника на функцію порівняння елементів масиву;
* функція порівняння елементів масиву для сортування на зростання (парні варіанти) або на спадання (непарні варіанти);
* функція порівняння елементів масиву для сортування за ускладненим правилом;
* функція вставки елементу до масиву з використанням вказівника на функцію. Реалізацію вставки робити шляхом послідовного перегляду елементів;

У методі main() мають послідовно вирішуватися такі завдання з виведення результату:

* введення масиву;
* сортування на зростання або спадання в залежності від варіанту;
* сортування за ускладненим правилом;
* вставка додаткового елементу.

Кількість елементів у масиві та параметри масиву має задавати користувач.

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* Тексти функцій для індивідуальних завдань з коментарями.
* Результати тестування проекту у вигляді копій консолі.
* Висновки.

Контрольні питання

* Сортування за методом вибору.
* Сортування за методом обміну.
* Сортування за методом вставкою.
* Сортування за ускладненими правилами.

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

Обробка впорядкованих масивів

Мета роботи:

* Ознайомитися з операціями над впорядкованими масивами.
* Створити програму, яка впорядковує та обробляє масив відповідно до індивідуального завдання.

Обробка впорядкованих масивів

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

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

Нижче розглядаються функції, які вирішують ці завдання.

### Пошук позиції елемента у впорядкованому масиві

Функція, яка повертає номер позиції елементу у впорядкованому масиві наведена на рисунку 12.1. Якщо елемент не знайдено, функція повертає -1.



Рисунок 12.1 – Функція пошуку позиції елемента у впорядкованому масиві

У наведеній функції для пошуку позиції елементу 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. 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.

### Вставка елементу до впорядкованого масиву

Алгоритм вставки елементу до впорядкованого масиву вже розглядався, як частина алгоритму сортування вставкою. Він полягає у послідовному аналізі елементів масиву, починаючи з останнього. Якщо 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.

Функція вставки елемента до впорядкованого масиву таким методом наведена на рисунку 12.2.

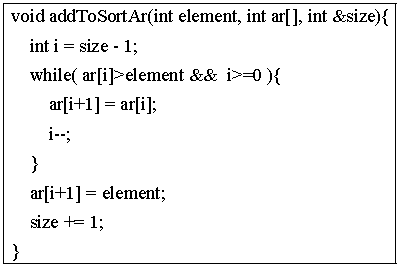


Рисунок 12.2 – Функція вставки елементу до впорядкованого масиву

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

Такий варіант функції вставки наведено на рисунку 12.3. Як бачимо, спроба підвищити ефективність алгоритму приводить до його ускладнення.



Рисунок 12.3 – Функція вставки елементу до впорядкованого масиву, яка використовує метод дихотомії

### Видалення елементу з упорядкованого масиву

Функцію видалення елементу з упорядкованого масиву наведено на рисунку 12.4.

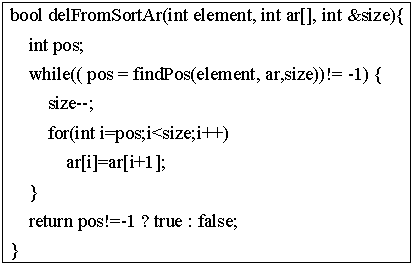


Рисунок 12.4 – Функція видалення елементу із впорядкованого масиву

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

У функції передбачено, що у масиві можуть бути однакові елементи.

### Злиття двох впорядкованих масивів

Функція злиття масивів наведена на рисунку 12.5.

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 циклу for, у якому одночасно змінюються обидва індекси і в якості перших індексів використовуються ті самі змінні і їх останні значення у попередньому циклі.

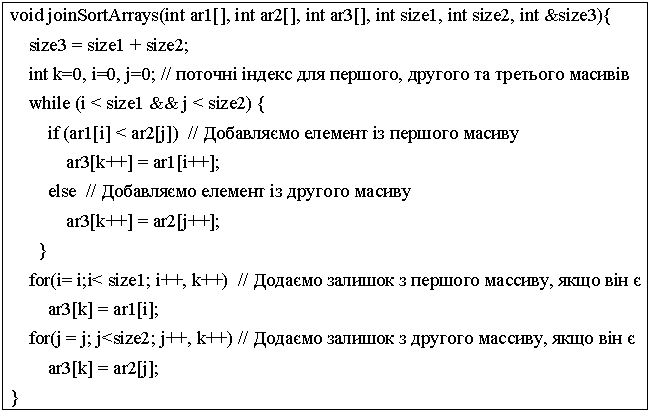


Рисунок 12.5 – Функція видалення елементу із впорядкованого масиву

Завдання для роботи

В лабораторній роботі слід створити програму, у якій будуть реалізовані алгоритми обробки упорядкованих масивів, що розглянуті вище.

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* Тексти функцій з коментарями.
* Результати тестування проекту у вигляді копій консолі.
* Висновки.

Контрольні питання

* Пошук елементу у впорядкованому масиві.
* Вставка елементу до впорядкованого масиву.
* Видалення елементу з впорядкованого масиву.
* Об’єднання впорядкованих масивів.

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432с.

структури

Мета роботи:

* Ознайомитися із поняттям структура.
* Ознайомитися із способами оголошення та ініціалізації структур.
* Опанувати алгоритми обробки структур.

Короткі теоретичні відомості

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.

### Оголошення шаблону та ініціалізація структур

Синтаксис оголошення шаблону структури виглядає так, як показано на рисунку 13.1.



Рисунок 13.1 – Синтаксис оголошення шаблону структури

На цьому рисунку:

* struct – службове слово, що використовується для визначення структур;
* тег шаблону структури – ім’я, яким позначають структури даної конструкції, фактично відіграє роль типу структури, хоча поняття тип не використовується;
* { } – дужки, що обмежують перелік полів структури;
* тип поля та ім’я поля – стандартне оголошення кожного з полів.

В якості прикладу розглянемо шаблон структури для збереження результатів атестації студентів. Структура буде містити чотири поля - прізвище студента з ініціалами, назва групи, в якій навчається студент, середній бал поточної успішності та кількість незадовільних оцінок.

Нижче наведено цей шаблон.

struct Stud {

char fio[20];

char gr[10];

float srBall;

int nzd;

};

Оголосити змінні, що мають відповідну структуру можна так:

struct Stud s1, s2, s3;

Слово struct у оголошенні таких змінних використовувати не обов’язково:

Stud s1, s2, s3;

Змінні можна оголошувати і одночасно із оголошенням шаблону:

struct Stud {

char fio[20];

char gr[10];

float srBall;

int nzd;

} s1, s2, s3;

При оголошенні структур їх можна ініціалізувати. Список полів обмежується фігурними дужками, а поля розділяються комами:

Stud bestStud = {“Гетьман Т.Г.”, “КІ-131”, 5.00, 0};

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, "Mазепа", 20);

Полями структури можуть бути також і структури.

### Масиви структур

Із структур однакового тегу може бути створений масив, точно так само як з даних інших типів. Наприклад, якщо ми хочемо мати результати атестації для всіх студентів, можна створити масив, що містить дані про кожного студента у вигляді структури і частково його проініціалізувати:

#define FIO\_SIZE 15

#define GR\_SIZE 7

#define AR\_SIZE 30

struct Stud {

char fio[FIO\_SIZE]; char gr[GR\_SIZE];

float srBall; int nzd;

};

Stud ar[AR\_SIZE]={

{”Чуб П.П.”,”КС-051”, 4.55, 0},

{”Гай А.Л.”,”КС-052”, 1.55, 3},

{”Кит А.В.”,”КС-051”, 2.45, 1},

{”Кiт 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\_SIZE);

strncpy(ar[size].gr, "КС-041", GR\_SIZE);

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’;

### Введення-виведення структур

При організації вводу-виводу варто мати на увазі, що безпосередньо структуру ввести або вивести, без використання спеціально написаних процедур, не можна. Можна вводити або виводити окремі поля структури, або елементи полів, якщо поля є теж структурами.

Введення і виведення структур схоже на введення та виведення матриць, але різниця тут у тому, що стовпчики можуть мати різні типи, і доступ до елементів різних стовпців проводиться не за індексом, а за назвою.

### 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 порівнянняння, які будуть передаватися у функцію сортування як параметр.

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 самостійної роботи.

### Визначення глобальних типів даних програми

Для реалізації проекту потрібно оголосити константи для розмірів полів структури та сам шаблон структури. Треба оголосити також посилання на масив структур, змінну, що буде зберігати його розмір, та змінну, яка буде зберігати кількість записів у масиві.

Ці оголошення повинні бути доступні усім функціям програми, тому їх слід зробити поза цими функціями, на початку файлу програми..

Тексти цих оголошень і ініціалізація масиву наведені нижче:

#define FIO\_SIZE 15  
#define GR\_SIZE 7  
  
struct Stud {char fio[FIO\_SIZE];  
 char gr[GR\_SIZE];  
 int srBall;  
 int nezd;  
};

Stud \*ar; // Посилання на масив структур

int capacuty; // Допустима кількість елементів у масиві

int size; // Кількість записів у масиві

### Початкова ініціалізація масиву

Головну функція проекту main() ми почнемо з ініціалізації змінних capacity та size і занесення відповідної кількості елементів до масиву.

Початкову ініціалізацію масиву ми робимо з метою спрощення тестування проекту і виправлення помилок. Спочатку створюємо масив у стеку а потім переписуємо його у динамічну пам’ять. Далі, у процесі додавання нових записів, capacity доведеться збільшувати, тому масив зберігаємо у динамічній пам’яті.

int main(){

. . .

Stud initAr [] = {  
 {“Чуб П.П.”, ”КС051”, 95, 0},  
 {“Гай А.Л. ”, ”КС052”, 35, 3},  
 {“Кит А.В. ”, ”КС051”, 55, 1},  
 {“Кiт С.В. ”, ”КС052»” , 25, 3},  
 {“Бутковський К.Л. ”, ”КС052»”, 85, 0}  
 };

capacuty = 6; // Стартова допустима кількість елементів = 6

size = 5; // стартова кількість записів у масиві = 5

// Виділення динамічної пам’яті для масиву

ar = new Stud[capacity];

//Копіювання масиву

memcpy(ar, initAr, size \* sizeof(Stud));

### Інтерфейс користувача для проекту

Формування інтерфейсу користувача також реалізуймо у головній функції програми, функції main().

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 може виглядати так:

typedef void MenuFunc();

struct MenuUnit{

string text;

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 використовується тип string для поля text, що зберігає посилання на текст пункту меню. Це дещо спростить роботу з відповідними текстами.

Ці оголошення можна розмістити у функції main(), бо вони більше ніде потрібні не будуть.

Можливий варіант інтерфейсу користувача представлено на рисунку 13.2.

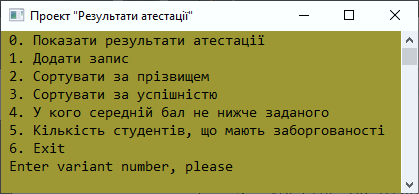


Рисунок 13.2 – Інтерфейс користувача проекту

Відповідно до цього рисунку створімо і одразу проініціалізуємо масив структур для пунктів меню:

MenuUnit menu[] = {

{"Показати результати атестацiї", showAll},

{"Додати запис", addRecord},

{"Сортувати за прiзвищем", sortByName},

{"Сортувати за успішністю", sortByResult},

{"У кого середнiй бал не нижче заданого",showForBall},

{"Кількість студентів, що мають заборгованості",

calculateProblem},

{"Exit", NULL},

};

Кожен елемент цього масиву має у своєму складі посилання на функцію (її ім’я). Тому слід оголосити прототипи цих функцій у загальній частині.

void showAll();

void addRecord();

void sortByName();

void sortByResult();

void showForBall();

void calculateProblem();

Окрім того, потрібно мати реалізацію цих функцій, бо інакше наше меню не спрацює. То ж доведеться 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. Наприклад, так (для функції showAll):

void pause(){

cout << "Для продовження натиснiть будь яку клавiшу" ;

getch(); // Потребує #include < conio.h>

}

/////////////////////////////////////////////////////////////////

void showAll(){

cout << " Call function showAll()" << endl;

pause();

};

У цьому фрагменті коду ще додано функцію pause(), якою ми часто будемо користуватися. Слід прописати її і у прототипах.

Тепер можна програмувати роботу меню.

Виведення варіантів роботи програми, введення номеру вибраного варіанту та виклик вибраної функції доцільно реалізувати у безкінечному циклі з очисткою екрану перед виведенням варіантів.

Продовжимо код функції main().

// Menu running

SetConsoleTitleA("Проект \"Результати атестацiї\"");

while(true){

system("cls");

int count = 0;

//Виводимо меню

for( ; ; count++){

cout << " " << count + 1 << ". "

<< menu[count].text << endl;

if(menu[count].text.compare("Exit") == 0)

break;

}

// Вводимо номер варiанту

int variant = 0; string s; bool ok = false;

while(variant < 1 || variant > count +1){

cout << " Enter variant number, please ";

getline(cin,s);

variant = atoi(s.c\_str());

}

if(variant == count +1){

cout << "\n Роботу з програмою закiнчено.\n";

break;

}

// Виклик функцiї

menu[variant - 1].func();

}

Після реалізації цього фрагменту можна запустити нашу програму і перевірити, чи працює наше меню.

Далі можна зайнятися реалізацією функцій меню.

### 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\_SIZE, "Студент",

GR\_SIZE, "Група", "Бал", "Борги");

for(int i=0;i<size;i++){

printf(" %-\*s %-\*s %3d %3d\n", FIO\_SIZE, ar[i].fio,

GR\_SIZE, ar[i].gr, ar[i].srBall, ar[i].nezd);

}

pause();

}

### Додавання нових даних до масиву структур

Додати нового студента у масив структур дуже просто. Для цього достатньо ввести з консолі значення кожного поля наступного елементу масиву і збільшити лічильник елементів масиву на 1.

Але ця функція, перш за все, має контролювати наповненість масиву. Якщо масив повністю заповнений, то потрібно виділити для нього нову ділянку пам’яті більшого розміру і переписати елементи повного масиву до нового.

Доцільно написати спеціальну функцію, яка буде виконувати таку роботу. Не забутьте оголосити прототип цієї функції.

void growAr(){

capacity = capacity + capacity / 2;

Stud\* newAr = new Stud[capacity];

// Copy old array to new

memcpy(newAr, ar, size \* sizeof(Stud));

Stud \*oldAr = ar;

ar = newAr;

// Old Deleting

delete[] oldAr;

}

Тепер можна написати функцію, що забезпечить додавання нового елементу до масиву.

Але тут можна зіштовхнутися із особливостями використання об’єкту cin та функції getline під час введення рядків символів та чисел. З цими проблемами ми вже зустрічалися у деяких попередніх роботах.

Окрім того, дані мають задовольняти деяким обмеженням.

Наступний код, сподіваємось, допоможе уникнути проблем введення даних.

void addRecord(){

if(size == capacity) growAr();

string s;

do{

cout<<"Введiть прiзвище: ";

getline(cin,s );

strncpy(ar[size].fio, s.c\_str(), FIO\_SIZE);

} while(strlen(ar[size].fio) == 0);

do{

cout << "Введiть групу: ";

getline(cin,s );

strncpy(ar[size].gr, s.c\_str(), GR\_SIZE);

} while(strlen(ar[size].gr) == 0);

int num; bool ok = false;

do{

cout <<"Введiть середнiй бал 0..100: ";

getline(cin,s);

num = atoi(s.c\_str());

} while(num <0 || num > 100);

ar[size].srBall = num;

do{

cout<<"Введiть кiлькiсть боргiв 0..10: ";

getline(cin,s);

num = atoi(s.c\_str());

} while(num <0 || num > 10);

ar[size].nezd = num;

size++;

}

### 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. Додаймо таке оголошення до загальної частини програми.

typedef bool 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 сортування.

Скористаймося для сортування алгоритмом вибору, як найпростішим:

void sort(Comparator test){

for(int i=0; i < size - 1; i++)

for(int j = i + 1; j < size; j++)

if ((!test(ar[i], ar[j]))){

Stud x = ar[i];

ar[i] = ar[j];

ar[j] = x;

}

}

### 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:

bool testByName(Stud s1, Stud s2){

return strcmp(s1.fio, s2.fio) <= 0;

}

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:

void sortByName(){

sort(testByName);

}

### 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:

bool testByResult(Stud s1, Stud s2){

if(s1.nezd != s2.nezd)

return s1.nezd < s2.nezd;

if(s1.srBall != s2.srBall)

return s1.srBall > s2.srBall;

return strcmp(s1.fio, s2.fio) <= 0;

}

А функція реалізації відповідного пункту меню буде такою:

void sortByResult(){

sort(testByResult);

}

### Вибірка студентів, що мають середній бал не нижче заданого

Вирішення цієї задачі мало чим відрізняється від задачі виведення масиву на консоль. Різниця полягає у тому, що потрібно організувати діалог з користувачем і виводити тільки ті структури, що відповідають заданій умові. Окрім того, масив перед обробкою доцільно відсортувати за результатами навчання.

void showForBall(){

int ball;

cout << " Введіть граничний бал ";

cin>>ball;

sort(testByResult);

cout << "\n Cтуденти, що мають бал не нижче " << ball << ":\n" ;

printf(" %-\*s %-\*s %s %s\n", FIO\_SIZE, "Студент",

GR\_SIZE, "Група", "Бал", "Борги");

for(int i=0;i<size;i++){

if(ar[i].srBall >= ball)

printf(" %-\*s %-\*s %3d %3d\n", FIO\_SIZE, ar[i].fio,

GR\_SIZE, ar[i].gr, ar[i].srBall, ar[i].nezd);

}

pause();

}

### Виведення інформації про кількість студентів що мають заборгованості

Вирішення цієї задачі потребує підрахунку кількості елементів у масиві, що відповідають заданій умові.

Напишемо таку функцію:

int calcDebt(){

int debtCount = 0;

for(int i = 0; i < size; i++)

if(ar[i].nezd > 0)

debtCount++;

return debtCount;

}

У реалізації функції меню ми окрім виведення кількості боржників ще надрукуємо інформацію про таких студентів у впорядкованому вигляді, якщо такі є.

void calculateProblem(){

int count = calcDebt();

if(count == 0)

cout << "\nСтудентiв iз заборгованостями нема." << endl;

else{

sort(testByResult);

cout << "\n Усього студентiв iз боргами "

<< count << ":\n" ;

printf(" %-\*s %-\*s %s %s\n", FIO\_SIZE, "Студент",

GR\_SIZE, "Група", "Бал", "Борги");

for(int i=0;i<size;i++)

if(ar[i].nezd > 0)

printf(" %-\*s %-\*s %3d %3d\n",

FIO\_SIZE, ar[i].fio,

GR\_SIZE, ar[i].gr,

ar[i].srBall, ar[i].nezd);

}

pause();

}

Завдання для самостійної роботи

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. Поля мають бути різних типів.

У проекті слід реалізувати такі функції:

* перегляд масиву;
* додавання нового елементу;
* вилучення елементу (вивести пронумерований перелік прізвищ, ввести потрібний номер, зробити зсув елементів і зменшити size);
* декілька варіантів сортування. Для сортування використовувати метод злиття;
* декілька варіантів вибору інформації із масиву.

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* Тексти функцій для індивідуальних завдань з коментарями.
* Результати тестування проекту у вигляді копій консолі.
* Висновки.

Контрольні питання

* Оголошення структури та її ініціалізація.
* Особливості масивів структур та їх ініціалізації.
* Структури як параметри функцій.
* Оголосити та ініціалізувати структуру за вказівкою викладача.
* Оголосити та ініціалізувати структуру відповідно до одного із варіантів індивідуальних завдань до лабораторної роботи.
* Написати функцію для обробки масиву структур.

## *Рекомендована література*

1. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 43

файли

Мета роботи:

* Ознайомитися з операціями над файлами.
* Створити програму, яка створює та обробляє файл.

Короткі теоретичні відомості

### Відкриття та закриття файлу

Файл – це послідовність байтів, яка зберігається на деякому зовнішньому носії інформації. 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.

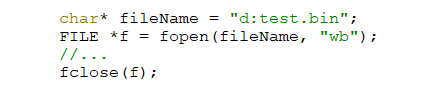
Ім’я файлу, що передається функції fopen() може бути повним, або скороченим (без адреси директорії). В разі передачі скороченого імені використовується активна на даний момент директорія.

Режим роботи встановлюється такими специфікаторами, таблиця 14.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 14.1 - 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 специфікатору 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 параметру передають вказівник на структуру, що пов’язана з файлом. Нижче наведено приклад відкриття і закриття файлу.



Після виконання цього фрагменту коду на диску d має з’явитися пустий файл test.bin.

### 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..

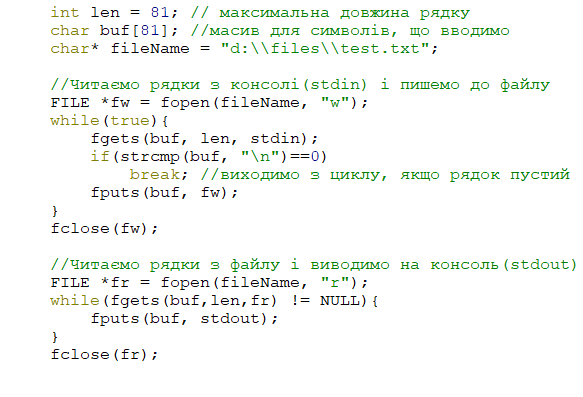
До функції fgets() в якості першого параметру передається вказівник на масив символів, куди буде записано прочитаний рядок. Другим параметром є число, що визначає максимально допустиму кількість прочитаних символів.

Слід зауважити, що із застосуванням перелічених функцій пов’язано багато нюансів. Докладніше про їх використання можна почитати у підручнику, або у довідкових джерелах інформації.

В якості прикладу використання цих функцій наводимо програму, яка спочатку зчитує рядки символів з консолі і записує ці рядки у файл. Завершується ця частина програми після того, як буде введено пустий рядок.

Після завершення введення даних файл закривається, а потім відкривається для читання, 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.



### 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 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 елементу блоку.

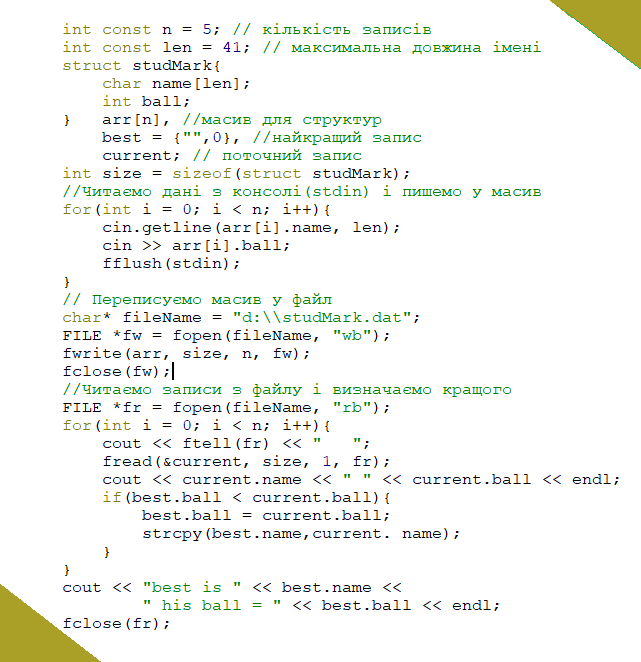
Третім параметром є кількість елементів у блоці.

Як приклад використання цих функцій наведемо код обробки даних про бали студентів. Структура studMark містить інформацію про ім’я студента та його бал успішності.

Дані про студентів заносяться з консолі у масив структур фіксованого розміру. Виклик функції fflush(stdin) потрібен для очищення буферу вводу, у якому залишається символ переходу на новий рядок після використання об’єкту cin для введення балу успішності.

Далі масив структур перепсується у файл і файл закривається.

На наступному етапі файл відкривається для зчитування і з нього послідовно зчитуються записи про кожного студента і знаходиться запис з найкращим балом.



### 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 параметру передається файлова змінна. Можна також скористатися функцією fgetpos(), але ця функція записує позицію у змінну, адреса якої передається в якості параметру.

За допомогою функції fsetpos() можна встановити значення позиції відповідно до значення змінної, адреса якої передається в якості параметру.

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 base визначає значення бази, яка розглядається як умовний 0. Можливі значення цього параметру наведені у табличці 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 |

Параметр offset визначає відстань позиції файлу від бази. Значення може бути як від’ємним так і додатнім.

В якості прикладу використання перелічених функцій розглянемо приклад сортування записів файлу методом бульбашки, використовуючи алгоритм розглянутий у лабораторній роботі 12.

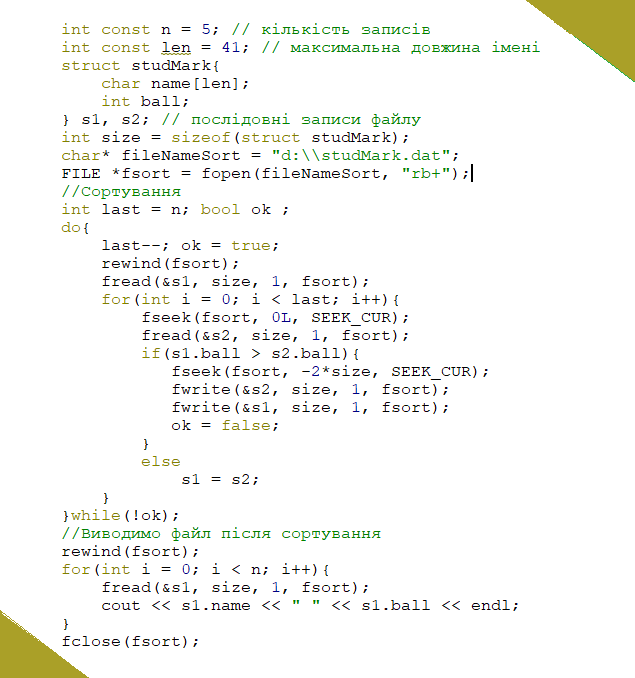
У програмі аналізуються два послідовних записи файлу, починаючи з початку. Запис, який був другим на якомусь етапі стає першим у наступному порівнянні. Якщо порядок розташування пари записів не відповідає умовам сортування, записи міняються місцями. Для переходу у потрібну позицію використовується функція 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 така 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.



Завдання для самостійної роботи

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:

* переписати масив у файл;
* переписати файл у масив;
* додати запис у кінець файлу;
* відсортувати файл;
* показати протокол змін.

Останній пункт передбачає наявність текстового файлу, у рядки якого записується інформація про додавання та вилучення записів масиву, або файлу.

Вимоги до звіту

* Назва роботи.
* Мета роботи.
* Тексти функцій для індивідуальних завдань з коментарями.
* Результати тестування проекту у вигляді копій консолі.
* Висновки.

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. Шпак З.Я. Програмування мовою С. – Львів: Оріяна-Нова, 2012. – 432 с.
2. Татарчук Д. Д., Діденко Ю. В. Програмування мовами С та С++: навч. посіб. / Д.Д. Татарчук, Ю.В. Діденко. – К.: , 2012. – 112 с.
3. Вінник В.Ю. Алгоритмічні мови та основи програмування мовою С. – Житомир: ЖДТУ, 2007.- 328 с.
4. Kernighan [B. W.,](http://www.amazon.com/Brian-W.-Kernighan/e/B000AQ1TNQ/ref=ntt_athr_dp_pel_1)  Ritchie D. M. C Programming Language / [Dennis M. Ritchie](http://www.amazon.com/s/ref=ntt_athr_dp_sr_2?_encoding=UTF8&field-author=Dennis%20M.%20Ritchie&search-alias=books&sort=relevancerank), [Brian W. Kernighan](http://www.amazon.com/Brian-W.-Kernighan/e/B000AQ1TNQ/ref=ntt_athr_dp_pel_1),  – 2nd ed. – Prentce-hall, inc., 1988. – 263 p.
5. Шилдт Герберт. Полный справ очник по С++. Пер. с англ. – М: Вильямс, 2004. 783 с.
6. C Programming Tutorial. – Доступу до ресурсу: <https://www.guru99.com/c-programming-tutorial.html>
7. Learn C and C++ Programming. – Доступу до ресурсу. : <https://www.cprogramming.com/>
8. 9 безкоштовних книг з мови Сі для початківців і не лише…(англійською мовою). – Доступу до ресурсу. : <https://echo.lviv.ua/dev/7366>
9. Методичні матеріали до курсу «Основи програмування». – Доступу до ресурсу: <https://eln.stu.cn.ua/login/index.php>