МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ІНФОРМАТИКА. ОСНОВИ ПРОГРАМУВАННЯ ТА АЛГОРИТМИ

МОВА ПРОГРАМУВАННЯ С. ЛАБОРАТОРНИЙ ПРАКТИКУМ

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітніми програмами «Інтелектуальні технології радіоелектронної техніки»,
«Інформаційна та комунікаційна радіоінженерія», «Радіотехнічні комп'ютеризовані системи», «Інформаційне забезпечення роботехнічних систем»
спеціальності
172 Телекомунікації та радіотехніки
126 Інформаційні системи та технології

Укладачі: С.В. Вишневий, П.Ю. Катін, Є.В. Крилов

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензенти

Корнага Я.І., докт. техн. наук, проф. кафедри інформаційних систем і технологій, ФІОТ, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Волокита А.М., канд. техн. наук, доц. кафедри обчислювальної техніки, ФІОТ, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Сушко І.О., канд. техн. наук, доц. кафедри прикладної радіоелектроніки, РТФ, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Відповідальний редактор

Вишневий С.В., канд. техн. наук

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 5 від 26.05.2022 р.)
за поданням Вченої ради Радіотехнічного факультету
(протокол № 04/2022 від 25.04.2022 р.)

В навчальному посібнику уміщено завдання на лабораторні роботи, приклади їх розв'язання та порядок оформлення результатів лабораторних робіт у процедурній парадигмі. Також посібник містить завдання домашньої контрольної роботи. Надано прототипи (шаблони) коду для використання у лабораторних роботах та додатковий інформаційний матеріал.

Для студентів всіх форм першого року навчання, які навчаються за спеціальністю 172 «Телекомунікації та радіотехніка» радіотехнічного факультету. Може бути використаний для навчання студентів за спеціальністю 126 «Інформаційні системи та технології» факультету інформатики та обчислювальної техніки.

Реєстр. № 21/22-400. Обсяг 6,3 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ВСТУП

Посібник містить завдання для лабораторних робіт (ЛР), рекомендації щодо їх виконання та порядку оформлення звітів. Завдання в посібнику орієнтовані на виконання студентами першого року навчання ступеня бакалавр.

Лабораторні роботи № 1–5 є обов'язковими для виконання і входять в блок основних (базових) завдань. Підготовлені студенти можуть виконати додатково ЛР № 6, 7. Студентам прискореного курсу навчання з досвідом в програмуванні пропонується ЛР № 8 для отримання додаткових балів, при умові здачі лабораторних робіт, що входять в блок основних завдань.

Лабораторні роботи № 1–4 складаються з двох частин. Перша частина є більш простим етапом і призначена для підготовки до другої частини лабораторної роботи. Студент, який виконав і захистив першу частину лабораторної роботи вважається таким, що захистив лабораторну роботу. Максимальна кількість балів за першу частину лабораторної роботи дорівнює половині від максимального значення, що нараховується за основне завдання відповідної лабораторну роботу. Виконавши першу частину лабораторної роботи, студент може виконати і захистити другу частину для підвищення оцінки.

Підготовлені студенти можуть виконувати відразу другу частину лабораторної роботи і в разі успішного захисту отримати відразу максимальну кількість балів без необхідності виконання першої частини лабораторної роботи.

Курс лабораторних робіт призначений для практичної підтримки навчальних дисциплін, що пов'язані із вивченням інших імперативних об'єктно-орієнтованих мов програмування, програмування мікроконтролерів, цифрової обробки сигналів та зображень, системного програмування та інших дисциплін, які пов’язані із необхідністю проведення моделювання при розробці та аналізі роботи радіотехнічних систем різного призначення.

Навчальний посібник може бути корисним також для спеціальності програмна інженерія на початковому рівні – при навчанні в бакалавратурі. Частина теоретичного матеріалу, програмного коду, рисунків, що наведені у посібнику, є авторськими, а інша частина — запозичена укладачами із різноманітних джерел. Зокрема, матеріал, що належить до основ алгоритмів запозичено з [1]. Базові підходи і теоретичний матеріал щодо мови програмування С з робіт [2, 3, 6, 8, 9]. Питання, які пов'язані з архітектурою комп'ютера з [4]. Матеріал, що стосується математичної теорії, запозичено з робіт [5, 7].

Основним завданням даного посібника є підготовка студентів до вивчення програмування на С–подібних мовах, до яких можна віднести C++, C#, Java, JavaScript. Крім того, знання і навички, що отримані у даному курсі лабораторних робіт, дають основу для розробки програмної частини радіотехнічних систем, програмування мікроконтролерів та компонентів вбудованих радіоелектронних систем.

Виконання лабораторних робіт у даному курсі дозволяється у різних інтегрованих середовищах розробки (IDE – Integrated Development Environment), які забезпечують зручний інструментарій для розробки програмного забезпечення на мові програмування С. Це зроблено з метою підтримання студентів, які мають практичний досвід програмування, для забезпечення можливості обрання того середовища розробки, яке буде найбільш зручним для відповідного студента.

Навчальний посібник розроблений для різних спеціальностей, які пов'язані з програмуванням. Передбачається можливість роботи з програмуванням мікроконтролерів, обробки сигналів для радіотехнічних систем різного роду тощо. Це обґрунтовує можливість використання різних компіляторів і стандартів мови С. Це веде того, що є імовірність наявності певних відмінностей у синтаксисі вихідного коду програм. У цьому випадку треба бути готовим, що певні зразки коду, що працювали на одних IDE не

будуть підтримуватися на інших. Для вирішення цього питання приведемо перелік стандартів мови програмування С:

- стандарт мови С ANSI X3.159-198 (С89), практично не відрізняється від стандарту ISO/IEC 9899: 1990 (С90) у цілому, різниця полягає лише у оформленні документації;
- стандарт мови С ISO/IEC (ICO/МЕК) 9899: 1990 (С90);
- стандарт мови С ISO/IEC 9899: 1999 (С99);
- стандарт мови С ISO / IEC 9899: 2011 (С11);
- сучасний стандарт - ICO/МЕК 9899: 2018 (С18). Стандарт доступний за URL http://www.open-std.org/JTC1/SC22/WG14.

Потрібно зазначити, що навіть у рамках одного стандарту мови програмування С можуть бути окремі особливості та відмінності в компіляторах, наприклад, компілятори для мікроконтролерів і компілятори для персональних комп'ютерів можуть в чомусь відрізнятися. Таким чином, розробнику треба бути готовим коригувати вихідний код відповідно до різних стандартів і компіляторів.

Основним IDE для виконання курсу лабораторних робіт є codeblocks-20.03 (дана версія є актуальною на момент написання посібника) з компілятором GCC, що призначений для операційних систем Windows 7/8.x/10. Крім того, є версії codeblocks-20.03 для Linux 32, 64-bit і для Mac OS. Таким чином, курс лабораторних робіт не прив'язаний до певної операційної системи.

Середовище розробки codeblocks-20.03 mingw-setup.exe побудовано на базі програмного проекту MinGW-W64 (версія 8.1.0, 32/64). У ньому забезпечується підтримка компілятора GCC/G++ та дебагеру GDB для операційних систем родини Windows.

Програмний проект MinGW-W64 підтримує розширення мови С для різних стандартів мови програмування. Деякі складові, що є частиною стандарту С99, приймаються як розширення в режимі С90. Окремі функції

мови і особливості синтаксису, які є частиною стандарту C11, приймаються як розширення у режимах C90 і C99.

За бажанням, можна обрати чітку версію стандарту мови програмування С. Це здійснюється за допомогою команд: std=gnu90 (для C90 з розширеннями GNU), std=gnu99 (для C99 з розширеннями GNU) або std=gnu11 (для C11 з розширеннями GNU). Більш докладний опис цих стандартів є відкритим і за потребою його можна знайти самостійно на веб-ресурсах відповідної тематики в Всесвітній мережі Інтернет.

Вибір мови програмування С для даного посібника для спеціальності 172 «Телекомунікації та радіотехніка» продиктовано широким її застосуванням при розробці програм для мікроконтролерів та мікропроцесорів, обробки сигналів тощо, які вивчаються у дисциплінах кафедри Радіотехнічних систем, а також у дисциплінах, які забезпечуються іншими кафедрами Радіотехнічного факультету. Мова програмування С активно використовується для розробки програмної складової вбудованих систем, а також сучасних радіотехнічних систем різного призначення. Крім того, вивчення мови програмування С, шляхом виконання відповідних навчальних завдань і лабораторних робіт, є основою для опанування інших подібних мов програмування. Прикладом цього є об’єктно-орієнтована мова програмування С++, що підтримує практично всі елементи мови С, і викладається на Радіотехнічному факультеті в рамках вивчення відповідної дисципліни. Таким чином, опанування мови програмування С та вироблення навичок з програмної реалізації відповідних алгоритмів, є основою для вивчення інших дисциплін, що базуються на даній дисципліні.

Дисципліна «Інформатика. Основи програмування та алгоритми» передбачає опанування мови програмування С через лабораторні дослідження, що дозволяють сформувати відповідні навички, для подальшої практичної чи наукової діяльності.

Основою навчального посібника є курс лабораторних робіт (ЛР № 1–5), що виконувався на Радіотехнічному факультеті у 2019-2021 році з першим курсом студентів під керівництвом к.т.н. Вишневого С.В. У навчальний посібник додані лабораторні роботи (ЛР № 6–7), що виконуються на факультеті інформатики та обчислювальної техніки у рамках дисципліни «Системне програмування на С і С++» 2020-2021, к.т.н. Катін П.Ю. Ці роботи є факультативними для підготовлених студентів.

Лабораторна робота № 8 призначена для студентів з досвідом роботи у галузі програмування, які навчаються за інтегрованими навчальними планами. Основою цієї ЛР є навчальна дисципліна «Методи і технології паралельного програмування», що викладалася к.т.н. Криловим Є.В. у 2019-2021 р.

Перша частина лабораторних робіт № 1–4, а також завдання для домашньої контрольної роботи, укладені Вишневим С.В. Друга частина лабораторних робіт № 1–4, лабораторна робота №7 укладені Катіним П. Ю.

Лабораторна робота № 5, вхідні і вихідні дані, завдання, приклади коду укладені Вишневим С.В. Теоретичні питання, щодо математичних положень до ЛР № 5 укладені Катіним П.Ю.

Лабораторні роботи № 6, 8, теоретичні відомості до ЛР № 3 укладені Криловим Є.В.

1. ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ ДЛЯ ЛАБОРАТОРНИХ РОБІТ

Для зручності і швидкого розроблення програм у теперішній час активно використовують інтегровані середовища розробки або Integrated Development Environment (IDE). Лабораторні роботи № 1–5 виконуються з використанням IDE. Звичайно IDE являє собою відносно складну програмну систему, що містить: базовий компілятор або інтерпретатор, редактор вихідного коду, інструменти для автоматизації компонування та налагодження програм та інші складові.

Однією з IDE, що є дуже зручною для навчання і може бути застосована у операційних системах (ОС) Windows, Linux і Mac OS X є середовище Code::Blocks. Це середовище розробки доступно на сайті проекту - codeblocks.org.

Крім того для виконання лабораторних робіт може бути використано середовище розробки QtCreator або інші, які дозволяють розробляти програмне забезпечення з використанням мови С.

Вихідний код, що представлений у навчальному посібнику, протестований з використанням Integrated Development Environment Qt Creator 4.13.3, побудований на бібліотеці Qt 5.15.2 (MSVC 2019, 64 біт). Збірка 2020 року, ревізія 524cad144a The Qt Company Ltd.

Лабораторні роботи № 7, 8 виконуються на основі компілятора GCC для операційних систем Linux і дають можливість підготуватися до роботи на системному рівні POSIX сумісних (сертифікованих) операційних систем.

2. ЛАБОРАТОРНА РОБОТА № 1 «РОЗРАХУНОК ТАБЛИЦІ МАТЕМАТИЧНОЇ ФУНКЦІЇ»

Мета роботи: полягає у набутті знань, умінь та навичок з технології розроблення програмного забезпечення (ПЗ) з використанням мови С у процедурній парадигмі. Надається досвід створення блок-схем алгоритмів.

Також лабораторна робота дає основні навички розробки з використанням IDE Code::Blocks. Дається можливість роботи з іншим типом IDE за вибором студента та по узгодженню з викладачем.

Перелік обладнання для виконання курсу лабораторних робіт № 1–5. Для підготовки ЛР № 1–5 може бути використаний будь-який сучасний ПК на основі мікропроцесора AMD64 (Intel® 64) або ARM. Операційна система Windows, Linux або Mac OS (Macintosh Operating System).

Лабораторна робота складається з 2-х частин. Підготовлені студенти можуть відразу виконувати 2-гу частину лабораторної роботи без зменшення кількості балів та без нарахування штрафних балів.

Перша частина лабораторної роботи передбачає виконання завдання для підготовки до другої частини лабораторної роботи. Рівень складності першої частини лабораторної роботи відповідає рівню складності лабораторної роботи № 1 описаної у [2]. Після виконання першої частини та її захисту, лабораторна робота вважається захищеною у цілому. За повне виконання та успішний захист першої частини лабораторної роботи нараховується половина від максимальної кількості балів, яка передбачена за виконання другої частини (основного завдання) лабораторної роботи.

Друга частина передбачає виконання основного завдання відповідно до силабусу даної дисципліни. За другу частину лабораторної роботи нараховується максимальна оцінка з урахуванням штрафних і додаткових балів, що передбачені за відповідний вид завдання.

Після перших 5-ти лабораторних робіт може бути обраний інший напрямок реалізації лабораторних робіт за вибором студента і за узгодженням з викладачем.

2.1 Частина перша. Типи даних та потоки вводу-виводу

Вхідні дані ЛР 1. Частина 1

Константні символьні рядки, що оголошені за допомогою директиви препроцесора *#define*, і кожен з яких зберігає: прізвище студента; ім'я студента; шифр навчальної групи; скорочену назву факультету; скорочену назву університету.

Змінні відповідних типів даних, кожна з яких зберігає наступну інформацію: вік студента; номер навчального семестру; вступний бал; оцінка за предмет № 1; оцінка за предмет № 2; оцінка за предмет № 3; середнє значення оцінки за три предмети.

Вихідні дані ЛР 1. Частина 1

Проект IDE Code::Blocks, вихідний код програми мовою С, що здійснює виведення на екран значень символьних констант; виведення на екран значень введених змінних та розрахованого значення середньої оцінки за предмети.

Завдання

1. Внести зміни в програму у розділі "*Теоретичні відомості ЛР 1. Частина 1*" , щоб на екрані виводилися власне ім'я та прізвище студента, який виконує лабораторну роботу, а також правильна абревіатура факультету та правильний шифр навчальної групи. Після очищення екрану, перед виведенням основної інформації, додати виведення на екран двох рядків, перший з яких складається з 15 символів * (символ «зірочка»), а другий рядок складається з

15 символів = (символ «дорівнює») (за бажанням можна обрати інші символи для формування рядків, які відділяють різні блоки даних на екрані монітора).

2. Внести зміни в програму у розділі "*Теоретичні відомості ЛР 1 Частина 1*", щоб значення змінної, що містить вступний бал (змінна *admissionScore*), виводилась би на екран з одним знаком після коми, а значення змінної *averageGrade* виводилась би на екран з двома знаками після коми.

3. Внести зміни в програму, що знаходиться у розділі "*Теоретичні відомості ЛР 1. Частина 1*", щоб задавалися значення оцінок п'яти (або більшої кількості) предметів. Зробити відповідні зміни у виразі для обчислення середнього балу (змінна *averageGrade*), щоб нараховувався середній бал за п'ять (або більшу кількість) предметів.

4. Створити акаунт на https://github.com/. Вихідний програмний код розмістити на сайті https://github.com/. Розмістити або у відкритому, або у приватному варіанті, із наданням прав доступу викладачу для первинної перевірки.

5. Звіт до першого частини лабораторної роботи оформити відповідно до вказаних вимог. Звіт також розташувати на https://github.com/.

6. Під час виконання першої частини ЛР1 потрібно навчитись працювати з IDE Code::Blocks. А саме, створювати проект, редагувати вихідний код, зберегати результат у вигляді вихідного коду, здійснювати компіляцію та виконання програми. Ознайомитись зі структурою вихідного коду на мові С.

Програма проведення експерименту ЛР 1. Частина 1

Розробка програми передбачає написання коду у текстовому редакторі та компіляцію цього коду у IDE Code::Blocks. Для розуміння процесу створення програми, наведемо у цілому послідовність дій, яку необхідно зробити для створення проекту і вихідного коду у вигляді наступної послідовності:

- провести первинний аналіз завдання, розробити формалізований опис задачі, обрати узагальнену моделі рішення;

- розробити алгоритм рішення завдання, алгоритм можна розробити у вигляді вихідного коду або блок-схеми алгоритму;
- провести проектування структури програми, для даного завдання — це рішення у вигляді одного файлу;
- створити проект у IDE Code::Blocks, зробити зміни у вихідному коді шляхом використання текстового редактора відповідно до завдання;
- зробити компіляцію і переконатися у правильній роботі програми;
- якщо вихідний код написаний без синтаксичних помилок, то компілятор середовища розробки Code::Blocks завершить свою роботу без повідомлень про помилки;
- при наявності помилок потрібно виправити їх і повторити компіляцію.

При відсутності помилок утворюється файл, наприклад hello (hello.exe для Windows). Цей файл (executable file) містить машинний код програми і додаткові елементи для запуску в операційній системі. Компілятор виконує перевірку вихідного коду на наявність синтаксичних помилок. У разі наявності помилок, він видає перелік цих помилок із посиланням на номер рядку, в якому допущена помилка.

Теоретичні відомості ЛР 1. Частина 1

Зазвичай вихідний код містить в собі наступні основні структурні частини:

- директиви препроцесора (*#include*, *#define*). Треба відмітити, що директиви препроцесора працюють до етапу компіляції програми, тобто на етапі її компонування;
- декларації зовнішніх функцій (сигнатури або прототипи);
- декларації (оголошення) глобальних змінних. Треба відмітити, що використання глобальних змінних доречно тільки у системному програмуванні, коли неможливо вирішити програмне завдання у інший

спосіб. Використання глобальних змінних у прикладному програмуванні потрібно, по-можливості, уникати;

- головну функцію (точку входу до програми) та її тіло у фігурних дужках *main()* {<*код програми*>};
- опис (реалізацію) зовнішніх функцій, сигнатури яких задекларовані до початку опису функції *main*(). Найпростіші програми можуть складатися лише з головної функції та директив препроцесора.

У мові програмування С директиви препроцесора *#include* використовуються для підключення бібліотек. Це відбувається з використанням заголовочних файлів з розширенням *.h*, наприклад *#include* <*stdio.h*>. Підключення відповідних заголовочних файлів дозволяє використовувати в програмі відповідний набір функцій стандартної бібліотеки, а також використовувати певні константи, що можуть бути визначені в окремих заголовочних файлах. Таким чином, при написанні програми потрібно визначити які саме бібліотечні функції можуть бути застосовані для вирішення окремих частин завдання при реалізації відповідних алгоритмів при виконанні лабораторних робіт.

При створенні нового проекту у IDE Code::Blocks по замовчуванню створюється файл вихідного коду, що показаний далі:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Для створення проекту треба активувати меню *Файл→Створити проект.* Результатом буде вікно програми, що показано на рис. 1 і містить варіанти проектів, що доступні у IDE Code::Blocks. Аналогічно працюють інші IDE у більшості випадків. Стрілкою на рис. 1 показано як обрати консольний проект для виконання лабораторної роботи № 1. Далі потрібно виконати всі етапи створення проекту, надати йому ім'я, визначити місця його розташування на локальному (або мережевому) диску.

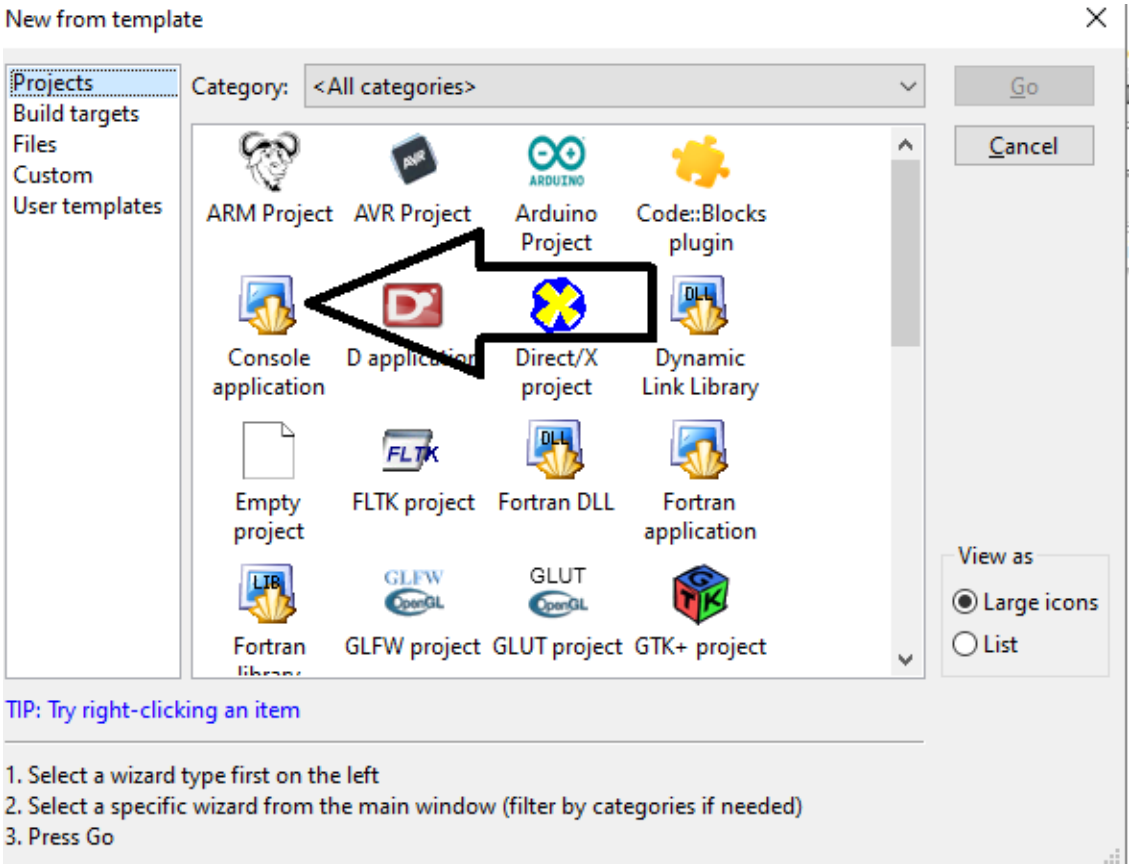


Рис. 1 — Вікно для вибору шаблонів додатків

Результатом є створення проекту і шаблонний вихідний код, що вказаний вище. Цей вихідний код програми містить основні структурні частини програми, директиви препроцесора (*#include*), головну функцію (точку входу до програми) та її тіло у фігурних дужках. Програма реалізує простий лінійний алгоритм.

Опис (реалізація) зовнішніх функцій також відсутня. Отже ця найпростіша програма складається лише з директив препроцесора, функції *main()* і виклику функцій *printf("Hello world!\n")*. Програма завершується оператором *return 0.*

Результатом правильного виконання є вікно IDE Code::Blocks, що зображено на рис. 2. У вікні представлений вихідний код, що являє собою приклад програми, що утворюється у IDE Code::Blocks по замовчуванню. У даному прикладі, робота лінійного алгоритму програми полягає у записі інформації на екрані вікна консолі (терміналу) з використанням функції *printf()*. Ця функція призначення для виведення інформації до потоку виводу. У консольній програмі цей стандартний потік виводу автоматично підключається до консолі (терміналу).

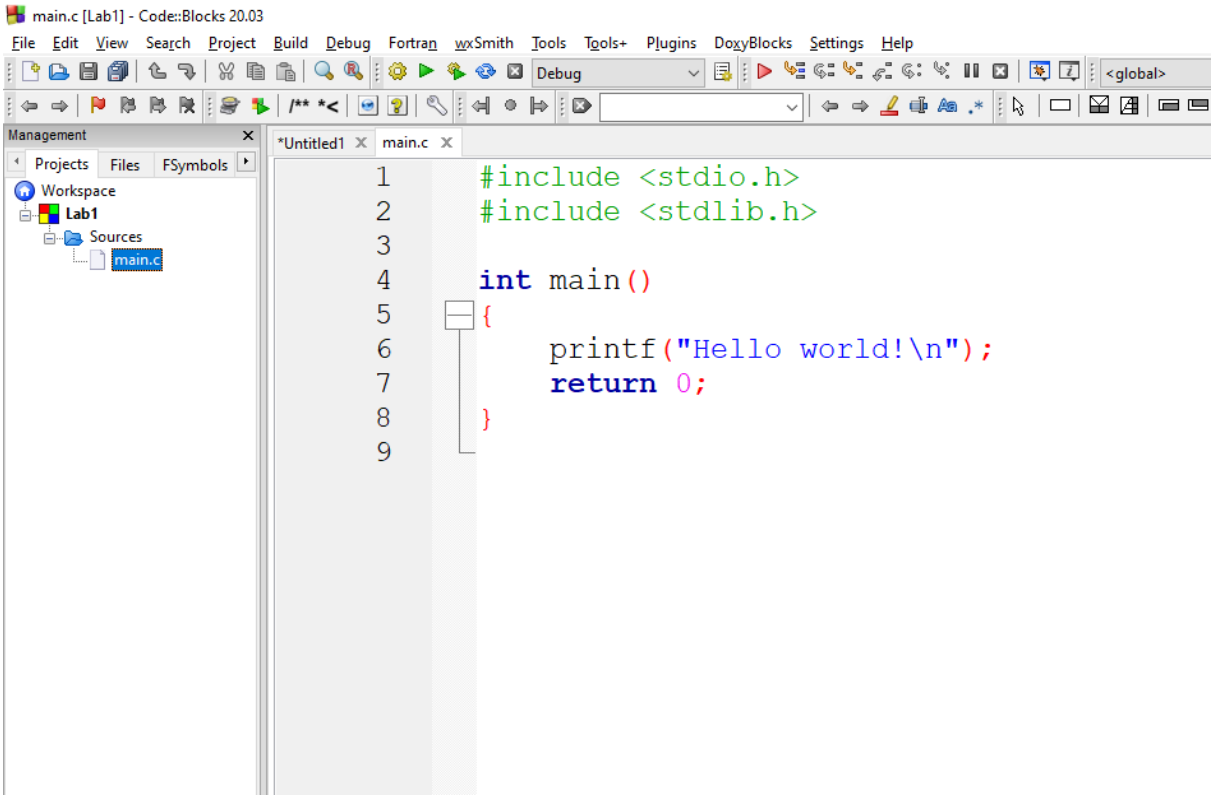


Рис. 2 — Вікно вихідного коду програми

Модель вводу-виводу у мові програмування С підтримується стандартною бібліотекою і дає можливість роботи з консольним вікном

(терміналом) або з файлами. Таким чином, процес текстового вводу чи виводу, незалежно від джерела і призначення, розглядається як робота з потоком знаків. Текстовий потік розглядається як послідовність знаків, розділених на рядки, де кожний рядок складається з символів, що завершуються знаком нового рядка.

Опишемо як працює функція *printf*(). Вона використовується для виведення інформації у стандартний потік виведення з управлінням формату. У наведеному на рис. 2 прикладі функція *printf*() записує інформацію в консоль (термінал) і має наступний узагальнений синтаксис виклику:

printf (<рядок-формату>,<елемент>,<елемент>,…);

Рядок формату починається та закінчується подвійними лапками. Приклад, що показаний на рис. 2, не містить додаткових змінних. Якщо потрібно вивести змінні на екран через *printf()*, ці змінні підставляють у рядок <елемент>,<елемент>,… Для правильного виведення змінних на екран потрібно використати відповідні команди форматування.

Команди форматування записуються до рядка *<рядок-формату>*. Приклад виведення нечислової змінної з використанням *printf()* показаний далі у фрагменті коду. Наведемо приклад використання функції, *printf("Приклад виведення цілої змінної %d \n", sum),* де *%d* – одна з команд форматування, що називається специфікатором формату. Всі специфікатори формату починаються із символу %. Після нього вказується буква, що задає тип даних або тип форматування виразу.

Кожному специфікатору формату відповідає рівно один елемент зі списку. Якщо тип даних елемента не відповідає специфікатору формату, можна отримати хибний результат виведення даних на екран. Елементами списку можуть бути змінні, константи, вирази, виклики функцій, тобто будь-яке значення, що відповідає доступному специфікаору формату. Прикладами специфікаторів формату можуть бути наступні:

%d — специфікатор формату цілого значення;

%u — специфікатор формату беззнакового цілого значення;

%ld — специфікатор формату довгого цілого значення;

%p — специфікатор формату значення вказівника;

%f — специфікатор формату числа з плаваючою крапкою;

%e — специфікатор формату числа з плаваючою крапкою з експонентою;

%c — специфікатор формату символьного значення;

%s — специфікатор формату для символьного рядка;

%x або *%X* — специфікатор формату цілого значення у шістнадцятковій системі числення.

Ширину поля виводу, у якому подаються дані, можна встановити за допомогою числа, що вказується між *%* та буквою специфікатора формату; наприклад, для виводу цілого значення в полі з шириною у 4 символи необхідно ввести *%4d*. Якщо необхідно вивести знак проценту, слід вказати *%%*. Крім символів специфікації формату використовуються символи, що мають вигляд *\n*. Ці спеціальні символи називають escape-послідовністю. Приклад *\n* означає переміщення курсору на початок нового рядка. Найбільш часто застосовують наступні escape-послідовності:

\f — переведення сторінки;

\t — табуляція;

\b — повернення курсору назад на одну позицію;

\x<hhh> — вставка символу, заданого ASCII-кодом *<hhh>*, де *<hhh>* шістнадцяткові цифри. Для використання функцій *scanf()* та *printf()* необхідно

за допомогою директиви препроцесора *#include* включити в програму бібліотечний заголовочний файл *stdio.h*. Це можна зробити таким чином:

```
#include <stdio.h>
```

Крім функції *printf()* використовують інші функції виведення, наприклад *puts()* та *putchar()*. Функція *puts()* виводить на екран рядок, додаючи до нього символ нового рядка. Наприклад, програму рис. 2 можна переписати у наступному вигляді:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  puts("Hello world!");
    return 0;
}
```

Константі символьні рядки можна оголосити за допомогою директиви препроцесора *#define*. Наприклад, константний символьний рядок, який буде мати ім'я *Name*, що буде зберігати ім'я студента, може бути оголошений наступним чином:

```
#define Name ”Mykola”
```

Аналогічно, константний символьний рядок для збереження прізвища студента, може бути названий *LastName*, і його оголошення буде таким:

```
#define LastName ”Ivanenko”
```

Між директивою *#define*, ім'ям константного рядка та значенням константного рядка ставиться символ пробіл. Схожим чином можуть бути створені константні символьні рядки, які будуть зберігати шифр групи (константа на ім'я *GroupName*); скорочену назву факультету (константа на ім'я *Faculty*); скорочену назву університету (константа на ім'я *University*):

```
#define GroupName ”GP-13”
#define Faculty ”FFF”
#define University ”Igor Sikorsky KPI”
```

Для зберігання іншої інформації, яка буде вводитися з клавіатури, повинні використовуватися змінні, тип даних яких повинен обиратися виходячи із того, які числові величини (цілі чи дійсні) мають бути введені.

Наприклад, для зберігання віку студента, може бути використана змінна типу *unsigned int*. Ім’я змінної бажано обирати виходячи із тих міркувань, щоб ім’я відображало суть інформації, яка в даній змінній зберігається. Наприклад, для збереження віку студента, ім’я змінної може бути таким: *age*. Оскільки значення змінної буде вводитися з клавіатури, то в тексті програми необхідно оголосити цю змінну, але можна її не ініціалізувати (тобто, не надавати початкового значення, яке задається одразу при оголошенні). Приклад оголошення та ініціалізації приведено нижче:

```
unsigned int age;              // оголошення змінної
unsigned int age = 18;         // ініціалізація змінної
```

Аналогічно, для збереження іншої інформації, яка передбачена умовою завдання, можна використати наступні змінні:

```
unsigned int semestr;   // змінна для збереження номеру семестру
float admissionScore;   // змінна для збереження вступного балу
int grade1;     // оцінка по предмету №1
int grade2;     // оцінка по предмету №2
int grade3;     // оцінка по предмету №3
double averageGrade;    // змінна, середнє значення оцінки.
```

Для введення даних з клавіатури за допомогою функції *scanf()* або для виведення інформації на екран монітора за допомогою функції *printf()*,

необхідно використовувати наступні специфікатори формату, які дозволяють вводити чи виводити значення відповідних типів даних:

%s — введення/виведення символьного рядка (набору символів),

%d або *%i* — введення/виведення цілих десяткових чисел із знаком ,

%u — введення/виведення беззнакових цілих десяткових чисел,

%f — введення/виведення чисел типу float,

%lf — введення/виведення чисел типу double.

Процес введення даних з клавіатури передбачає виведення на екран супутньої інформації, яка б давала можливість користувачу зрозуміти що від користувача очікують. Наприклад:

```
printf(”%s %s виконайте наступні дії.\n”, Name, LastName);
printf(”Введіть номер семестру: ”);
scanf(”%u”, &semestr);

printf(”Введіть свій вступний бал: ”);
scanf(”%f”, &admissionScore);

printf(”Введіть оцінку за предмет №1: ”);
scanf(”%d”, &grade1);

printf(”Введіть оцінку за предмет №2: ”);
scanf(”%d”, &grade2);

printf(”Введіть оцінку за предмет №3: ”);
scanf(”%d”, &grade3);

printf(”Введіть свій вік: ”);
scanf(”%u”, &age).
```

На екрані буде виведено повідомлення:

Mykola Ivanenko виконайте наступні дії.

Введіть номер семестру: [очікування введення з клавіатури].

Наприклад, якщо при кожному виклику функції *scanf()*, користувач вводив відповідні дані, то вигляд консольного вікна, буде наступним:

Mykola Ivanenko виконайте наступні дії.

Введіть номер семестру: ***1***

Введіть свій вступний бал: ***162.5***

Введіть оцінку за предмет №1: ***65***

Введіть оцінку за предмет №2: ***79***

Введіть оцінку за предмет №3: ***95***

Введіть свій вік: ***18***

Відповідно до завдання необхідно розрахувати середній бал на основі введених оцінок. Для цього була оголошена змінна на ім'я *averageGrade* типу *double*, що використовується для збереження результату обрахунку середнього балу.

Враховуючи, що оцінки за предмет 1, 2 та 3 зберігаються в змінних типу *int*, то для виконання правильного підрахунку середнього балу, при якому буде збережена дробова частина, необхідно виконати явне перетворення типу даних:

```
averageGrade = (double)(grade1 + grade2 + grade3) / 3;
```

Для виведення на екран значення середньої оцінки із трьома знаками після коми, потрібно в специфікаторі формату вказати відповідне значення. Наприклад, функція *printf()* буде мати наступний вигляд:

```
printf("\n\t average grade = %.3lf", averageGrade);
```

Послідовність \n дозволяє перейти на новий рядок, і з нового рядка виводити результат.

Послідовність *t* дозволяє зробити відступ (табуляцію) і після цього виводити текст в консольне вікно. Для того, щоб мати змогу очистити командне вікно від тексту, що відображений в командному вікні, можна, наприклад, скористатися викликом наступної функції:

```
system(”cls”);
```

Для того, щоб мати змогу застосовувати функцію *system(”cls”)*, необхідно підключити заголовний файл *stdlib.h* за допомогою наступного запису:

```
#include <stdlib.h>
```

Після введених даних, необхідно очистити екран, і вивести в консольне вікно значення символьних констант, а також значення введеного вступного балу студента, а також розрахованого середнього балу по предметам, при цьому оформлення виводу може виконуватися за допомогою символів псевдографіки. Наприклад, текст на екрані може бути виведений таким чином:

```
Igor Sikorsky Kyiv Polytechnic Institute
Faculty: FFF
Group: GP-13
----------------------------
student: Mykola Ivanenko
age: 18
semestr: 1
----------------------------
admission score: 162.50
average grade: 79.667.
```

Текст програми може бути наступним (враховуючи, що при роботі з консольним вікном, виведення тексту, що написаний літерами українського алфавіту може відбуватися некоректно, можна використовувати запис

текстових повідомлень на трансліті (запис текстових повідомлень буквами англійського алфавіту), або ж записувати текстові коментарі англійською мовою):

```
#include <stdio.h>
#include <stdlib.h>

#define Name "Mykola"
#define LastName "Ivanenko"
#define GroupName "GP-13"
#define Faculty "FFF"
#define University "Igor Sikorsky Kyiv Polytechnic Institute"

int main()
{
    unsigned int semestr;
    float admissionScore;
    int grade1;
    int grade2;
    int grade3;
    double averageGrade;
    unsigned int age;

    printf("%s %s Enter following data.\n", Name, LastName);

    printf("Enter semestr:");
    scanf("%u", &semestr);

    printf("Enter your admission score: ");
    scanf("%f", &admissionScore);
```

```
    printf("Enter grade #1: ");
    scanf("%d", &grade1);

    printf("Enter grade #2:");
    scanf("%d", &grade2);

    printf("Enter grade #3:");
    scanf("%d", &grade3);

    printf("Enter your age: ");
    scanf("%u", &age);

    averageGrade = (double)(grade1 + grade2 + grade3) / 3;

    system("cls");
    printf("%s", University);
    printf("\n%s", Faculty);
    printf("\n%s", GroupName);
    printf("\n---------------------------");
    printf("\nstudent: %s %s", Name, LastName );
    printf("\nage: %u", age);
    printf("\nsemestr: %u", semestr);
    printf("\n---------------------------");
    printf("\nadmission score: %.2f", admissionScore);
    printf("\naverage grade = %.3lf", averageGrade);
    printf("\n");
    return 0;
}
```

Для використання змінної у мові програмування С її потрібно оголосити. Оголошення змінних виконується за наступним шаблоном:

<тип даних> <ім'я змінної> <= значення>;

При цьому для змінної виділяється оперативна пам'ять ПК і виконується ініціалізація змінної. Ініціалізацію змінної можна не робити відразу після оголошення. Наведемо приклади ініціалізації змінних:

```
int var1 = 177;

unsigned char var2 = '1';

float var3 = 3.1415;

char sym= 'A';
```

Існує кілька основних типів даних у С: *char*, *int*, *float*, *double* разом з модифікаторами *short*, *long*, *signed*, *unsigned*.

Наприклад, *int* означає, що відповідні змінні призначені для збереження цілих чисел. Тип даних *float* та *double* визначають у програмі змінні для роботи з числами з рухомою (плаваючою) крапкою (комою), тобто числа, що можуть мати дробову частину. Розмір змінних у байтах типу *int* і *float*, в деяких випадках залежать від типу ПК.

Особливість типів даних мови С полягає у тому, що ці типи даних прямо пов'язані з архітектурними елементами мікропроцесорів або мікроконтролерів. Наприклад, у процесорах архітектури AMD64 (Intel® 64), що найбільш розповсюджені на теперішній час як основа ПК, підтримуються апаратно беззнакові цілі числа, що зв'язані з беззнаковими цілочисловими типами даних мови програмування С, наприклад *unsigned int*. Перелік беззнакових цілих чисел (unsigned integer), що підтримуються у архітектурі AMD64 (Intel® 64), наступний [4]:

- 8-бітове (байтове) ціле число без знака,

- 16-бітове (слово) ціле без знака,
- 32-розрядне (doubleword) ціле число без знака,
- 64-розрядне (quadword) ціле число без знака,
- 128-розрядне (octword) ціле число без знака.

Крім того у процесорах архітектури AMD64 (Intel® 64) підтримуються апаратно також знакові цілі числа, signed integer [4]:

- 8-бітове (байтове) ціле число зі знаком,
- 16-бітове (слово) ціле число зі знаком,
- 32-розрядне (doubleword) ціле число зі знаком,
- 64-розрядне (quadword) ціле число зі знаком,
- 128-бітове (octword) ціле число зі знаком.

Узагальнений опис цілочислових даних процесорів архітектури AMD64 (Intel® 64) показаний на рис. 3 [4]. Особливість типу даних *float* (*double*) полягає у тому, що цей тип забезпечує роботу з числами у форматі з рухомою (плаваючою) крапкою (комою). Цей тип даних апаратно підтримується у сучасних процесорах AMD64 (Intel® 64), а саме:

- тип даних із плаваючою комою (half-precision floating point (16 bits),
- single-precision floating point (32 bits),
- double-precision floating point (64 bits).

Тобто, для реалізації типу даних мови програмування С *float* (*double*) у мікропроцесорах передбачений спеціальний набір регістрів [4].

В деяких випадках, у відносно простих мікроконтролерах або мікропроцесорах немає апаратної підтримки типу даних *float* (*double*). У цьому

випадку реалізація типу даних *float* (*double*) відбувається програмно, це суттєво сповільнює швидкість обчислень.

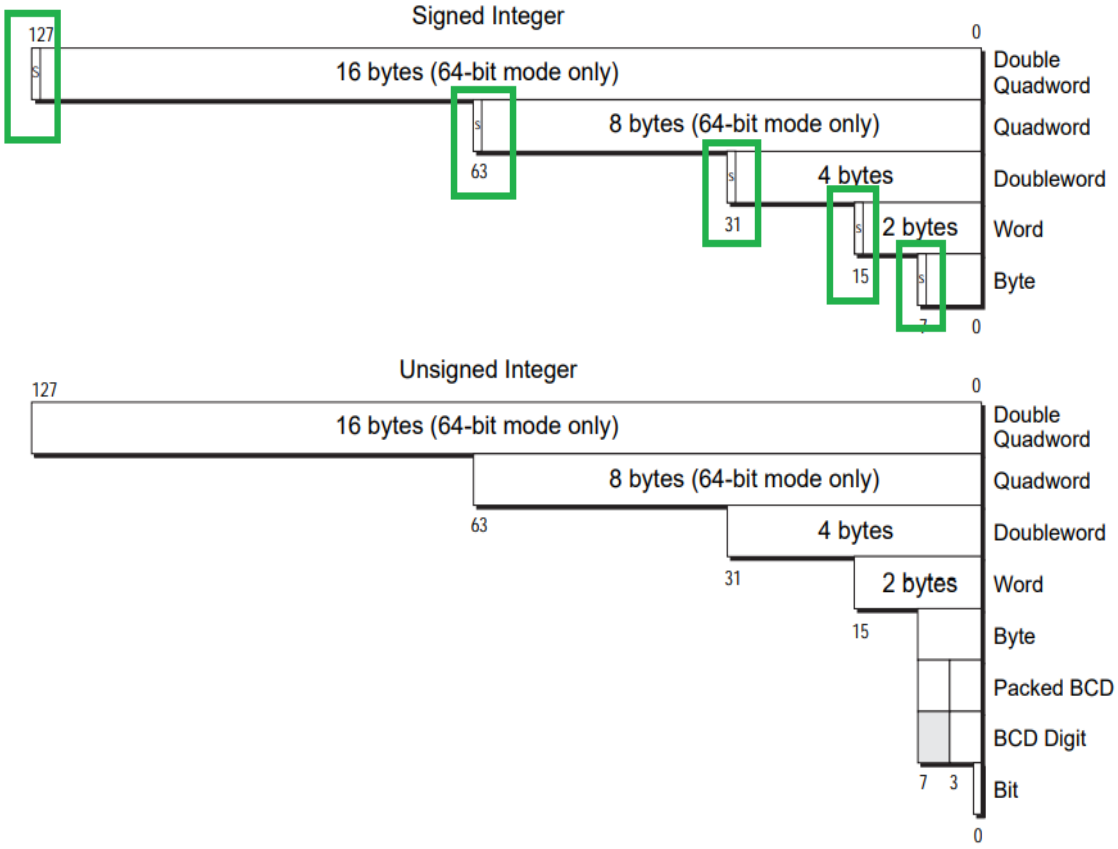


Рис. 3 — Механізм апаратної підтримки типів даних AMD64 (Intel® 64) [4]

Окрім цього, існують ще масиви, структури та сполуки з цих основних типів та вказівники на них.

2.2 Частина друга. Розрахунок таблиці функції

Вхідні дані ЛР 1. Частина 2

Вихідна функція $f(x)$, що обирається відповідно до варіанту з таблиці 1. Початкове та кінцеве значення аргументів для $f(x)$ вводяться з клавіатури. Також з клавіатури вводиться кількість точок у таблиці або крок зміни аргументу. Вихідна функція $f(x)$ буде у подальшому використовуватися для виконання ЛР 3.

Таблиця 1 — Варіанти завдань

Номер варіанту	Вихідна функція $f(x)$
1	$f(x) = x^4 - 5x^3 - 0.25x^2 + 2$
2	$f(x) = 0.25x^5 + 4(x+20)^4 - 2x$
3	$f(x) = -3x^6 - 10(x-30)^5 + 12x^4 + 5x^3 - 10$
4	$f(x) = \frac{x^3}{30} - 4x^2 + 50$
5	$f(x) = 0.4(x-100)^3 + 0.3x^2 - 0.2x + 200$
6	$f(x) = \frac{x^2}{1000} - 0.5x - 1000$
7	$f(x) = -0.5(x-64)^3 - 3x^2 + 10$
8	$f(x) = -4x^3 - 3x^2 + 2x^2 - 5x - 25$
9	$f(x) = 4\left(\frac{x}{10} - 2\right)^3 - 15x + 12$
10	$f(x) = 0.25(x-25)^2 + \frac{(x+25)^3}{100} + 1$
11	$f(x) = 0.5(x+250)^3 - (x-125)^2 + 500$

Таблиця 1 — Варіанти завдань (продовження)

Номер варіанту	Вихідна функція $f(x)$
12	$f(x)=\left(\frac{x}{100}-5\right)^5-\left(\frac{x}{50}+10\right)^4-\left(\frac{x}{25}-15\right)^3-x^2-10$
13	$f(x)=(x-10)^3\sin\left(\frac{x+10}{100}\right)-10x+12$
14	$f(x)=(x+50)^5\sin\left(\frac{x-5}{60}\right)-(x-15)^3-3x^2-20$
15	$f(x)=2(x-1000)^3-3(x+500)^2+4x-5$
16	$f(x)=(x-50)\cos\left(\frac{x}{10}\right)-3x+500$
17	$f(x)=(x+125)^3-45x-18$
18	$f(x)=\frac{x^3}{20}-5x^2+1000$
19	$f(x)=(x+100)\sin\left(\frac{x}{5}\right)-5(x-10)-2100$
20	$f(x)=(x-200)^5-(x-100)^3-50x-25$
21	$f(x)=8\left(\frac{x}{16}-4\right)^3-4x-12$
22	$f(x)=(x+1000)\sin^2\left(\frac{x}{50}\right)-8x+1800$

Таблиця 1 — Варіанти завдань (продовження)

Номер варіанту	Вихідна функція *f*(*x*)
23	$f(x) = 9x^3 + 1000$
24	$f(x) = 2x^2 - 3x^3 + 1500x - 12000$
25	$f(x) = -\frac{x^3}{25} + 10x^2 - 15000$

Вихідні дані ЛР 1. Частина 2

Проект IDE Code::Blocks та вихідний код програми мовою С. Таблиця аргументів функції *f*(*x*) та її значень (складається зі стовпчиків, що умовно можна назвати "*номер точки*", "*значення аргументу*" та "*значення функції*"). Функція *f*(*x*) для розрахунку таблиці обирається відповідно до варіанту.

Ця лабораторна робота призначена для підготовки даних для виконання лабораторної роботи № 3. За результатами ЛР № 1 треба визначити відрізки значень аргументів функції, де вона має різні знаки.

Завдання

1. У якості індивідуального завдання, на першому етапі написати код, що реалізує текстовий інтерфейс, у якому обирається один із двох можливих варіантів введення початкових даних.

1.1 Перший варіант введення початкових даних передбачає ведення наступних даних:

- початкове значення аргументу (*X1*),
- кінцеве значення аргументу (*X2*),

- кількість точок у таблиці (*N*).

В цьому випадку на основі введених початкових даних необхідно виконати розрахунок кроку зміни аргументу (*delta*).

1.2 Другий варіант введення початкових даних передбачає введення наступних даних:

- початкове значення аргументу (*X1*),
- кінцеве значення аргументу (*X2*),
- крок зміни аргументу (*delta*).

В цьому випадку на основі введених початкових даних можливо виконати розрахунок кількості точок у таблиці (*N*), щоб, наприклад, використовувати цей обрахований параметр для побудови таблиці.

2. Для збереження значень *X1, X2, delta* потрібно в програмі оголосити змінні, що мають тип даних *double*, а для збереження *N* оголосити змінну, що має цілий тип даних, наприклад, *unsigned int*.

3. У програмі передбачити валідацію введеного номеру варіанту задання початкових даних (контроль допустимого значення під час введення даних).

4. Перед виводом таблиці на екран необхідно вивести введені початкові дані. Таблицю накреслити за допомогою символів псевдографіки або інших символів (ширина кожного стовпця береться з таблиці 4 відповідно варіанту). Передбачити зупинку виведення таблиці при заповненні екрану, для продовження виведення натиснути будь-яку клавішу, при цьому таблиця продовжує виводитися до моменту заповнення екрану, після чого процес повторюється до закінчення виведення.

5. Приклад вигляду першого екрану (першої частини таблиці, яка повністю розміщується в межах ширини командного вікна) для випадку, коли вибраний 2-й варіант введення початкових даних, показаний на рис. 4.

6. Після натискання клавіші ENTER продовжити виведення таблиці. При цьому таблиця повинна бути суцільною, тобто після натискання ENTER надпис

Press Any Key to Continue... повинен зникнути і має продовжуватись виведення таблиці. Приклад вигляду продовження таблиці показаний на рис. 5.

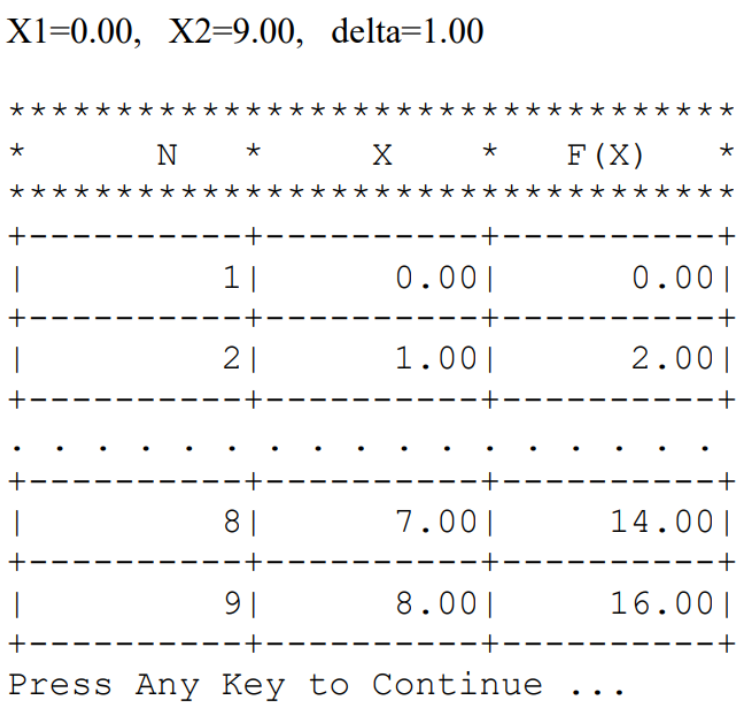


Рис. 4 — Приклад вигляду першого екрану

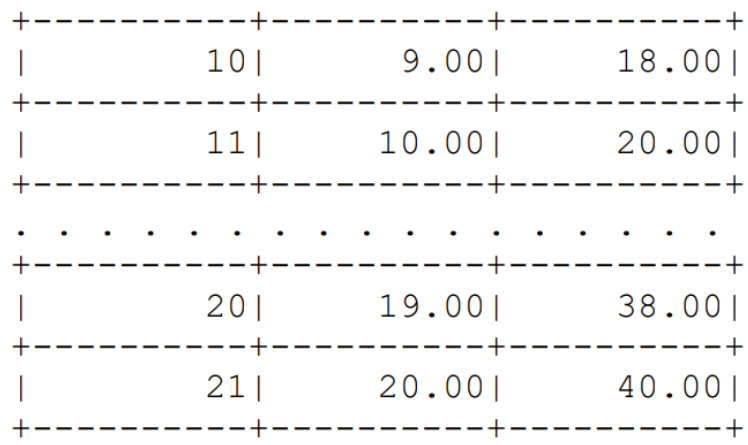


Рис. 5 — Вигляд другого (і наступних) екранів

7. При виведені залишка таблиці для другого і наступних екранів повідомлення *Press Any Key to Continue ...* не виводити на екран, але залишити

зупинку виведення таблиці до натискання будь-якої клавіші. Отже програма має виводити таблицю поступово до завершення.

8. Визначити діапазони аргументів у яких функція змінює знак на протилежний. Побудувати блок схему алгоритму програми.

9. Звіт до 2 етапу лабораторної роботи оформити відповідно до наданих дали вимог. Звіт також розташувати на https://github.com/.

Програма проведення експерименту ЛР 1. Частина 2

Далі наведений приклад програми проведення експерименту, що може бути частково змінено за рішенням студента.

1. Написати код, що реалізує текстовий інтерфейс, у якому обирається два варіанта введення початкових даних відповідно до завдання.

1.1 Перший варіант початкових даних передбачає введення:

- початкового значення аргументу (*X1*),
- кінцевого значення аргументу (*X2*),
- кількість точок в таблиці (*N*).

1.2 Другий варіант початкових даних передбачає введення:

- початкового значення аргументу (*X1*),
- кінцевого значення аргументу (*X2*),
- крок зміни аргументу (*delta*).

2. Доповнити код п.1, реалізувати валідацію варіанту (контроль допустимого значення під час введення даних). Налагодити і протестувати рішення.

3. Доповнити код п.2 реалізувавши розрахунок значень функції у діапазоні значення аргументу (*X1...X2*). Налагодити і протестувати рішення.

4. Доповнити код п.3, реалізувати вивід таблиці на екран відповідно до завдання. Налагодити і протестувати рішення.

5. Доповнити код п.4, реалізувати програмне визначення діапазонів аргументів у яких функція змінює знак на протилежний. Вивести їх на екран. Налагодити і протестувати рішення. Результат розташувати на https://github.com/.

6. Побудувати блок-схему алгоритму програми.

7. Оформити звіт лабораторної роботи відповідно до вказаних вимог. Звіт також розташувати на https://github.com/.

Теоретичні відомості ЛР 1. Частина 2

Результати даної лабораторної роботи призначені для виконання лабораторних робіт № 3 і 5. Функція, що задана у варіантах таблиці 1 буде використовуватися у ЛР 3 для розроблення програми рішення нелінійного рівняння. Позначимо нелінійне рівняння у загальному вигляді як

$$f(x) = 0 \qquad (1.1)$$

Значення змінної x при яких виконується рівняння (1.1) називають рішеннями нелінійного рівняння (1.1). Позначимо ці значення у наближеному вигляді як ξi (де $i = 1, 2, ...$), за яких виконується рівність $f(\xi i) = 0$. Ці значення називаються нулями функції $f(x)$ і розв'язками рівняння (1.1). Рівняння (1.1) може мати один або кілька розв'язків. Може бути такий випадок, що (1.1) не має жодного рішення.

Побудуємо приклад графіку функції $f(x)$ рівняння (1.1) на проміжку $[\alpha, \beta]$ на якому є рішення рівняння. Приклад наведено на рис. 6. Розв'язок зображується як точка перетинання функції $f(x)$ та осі x (абсцис). Існують багато чисельних методів знаходження рішення рівняння (1.1). У більшості випадків на першому кроці для рішення (1.1) потрібно знайти саме відрізок $[\alpha, \beta]$, приклад якого показаний на рис. 6. На цьому відрізку функція $f(x)$ змінює знак на протилежний. Саме тому потрібно знайти у лабораторному

дослідженні проміжки на яких вихідна функція змінює знак на протилежний з довільною точністю.

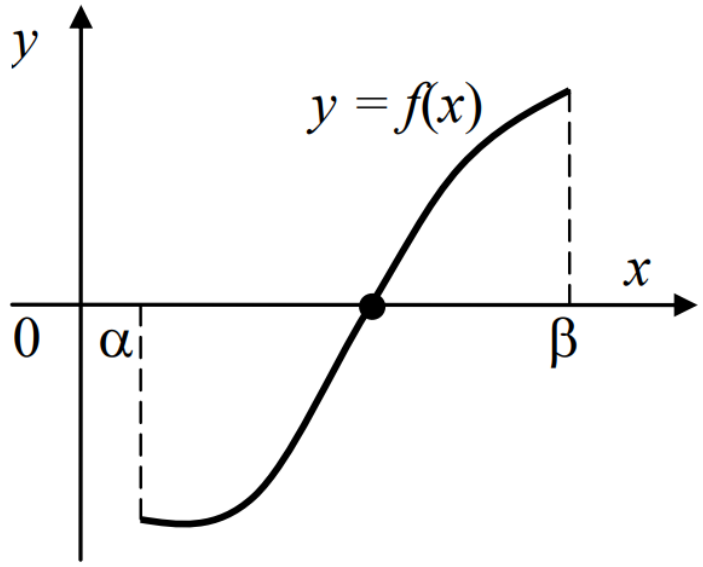


Рис. 6 — Геометрична інтерпретація розв’язків рівнянь на інтервалі [α, β] [5]

Проміжок [α, β], на якому є один і лише один розв’язок ξ рівняння (1.1), називають проміжком ізоляції, а сам розв’язок називають ізольованим на заданому проміжку. Отже, наближені методи розв’язування рівнянь, здебільшого складаються з двох етапів:

- визначення проміжків ізоляції з метою обчислення наближених значень розв’язків рівняння (у цьому разі говорять про початкове (чи нульове) наближення);

- обчислення значень розв’язків рівняння із заданою точністю. Для визначення всіх проміжків ізоляції достатньо обчислити таблицю значень $f(x)$. Саме це завдання виконується у ЛР1. Для розрахунку таблиці і знаходження проміжку [α, β] потрібно написати програму, що реалізує певний алгоритм.

Алгоритм – це однозначна послідовність дій, що необхідно виконати під час розв’язування певної задачі [1]. Існує певна кількість формальних варіантів, що дають можливість задавати сучасні алгоритми і виконувати їх документацію. Розглянемо традиційні способи визначення і документування

алгоритмів до яких належать наступні:

– у графічній формі, у вигляді блок-схеми, що являє собою графічне зображення алгоритму, зміст кожного кроку алгоритму записують у визначеній формі у вигляді блоку (геометричної фігури), порядок виконання кроків вказують за допомогою стрілок, що з'єднують блоки;

– спосіб задання алгоритму у вигляді псевдокоду. Псевдокод – це опис алгоритмів умовною алгоритмічною мовою, що включає елементи мови програмування, фрази природної мови, загальноприйняті математичні позначення тощо;

– спосіб задання алгоритму у вигляді комп'ютерної програми або вихідного коду (приклад на мові С), для задання алгоритму у вигляді програми використовують конструкції конкретної мови програмування. На теперішній час найбільш поширеним способом є спосіб задання алгоритму у вигляді комп'ютерної програми. Після розробки програми, алгоритм можна представити у вигляді блок-схеми або в іншому варіанті.

Можливий інший підхід до побудови алгоритмів, що частіше використовується математиками або іншими технічними спеціалістами. У цьому випадку спочатку треба розробити блок-схему, потім побудувати програму. Найбільш загальний варіант алгоритму передбачає виконання наступної послідовності окремих пунктів, а саме:

- отримання вхідних даних;
- обробка або перетворення вхідних даних під час виконання програми (або іншого завдання) і отримання вихідних даних (іншого результату);
- вивід вихідних даних (іншого результату) для користувача.

Розробка алгоритму передбачає розбиття первинного завдання на певні етапи, що виконуються послідовно, результати виконання попередніх етапів можуть використовуватися при виконанні наступних. При цьому має бути чітко визначений зміст кожного етапу і порядок виконання етапів. Окремий етап

алгоритму є частиною первинного завдання і повинен бути доволі простим і зрозумілим без пояснень.

У теперішній час відбувається продовження вивчення теорії алгоритмів, у зв’язку з проблемою неможливості розв'язання певних завдань алгоритмізації [1]. Це обумовлено тим, що при спробах розв'язування певних задач виникли питання, що залишаються невирішеними, незважаючи на зусилля математиків. Згодом дійшли до висновку, що існує коло задач, які практично не вирішуються алгоритмічно (так звані NP-повні задачі).

Для вирішення проблем, що виникають при розробці алгоритмів, зокрема, для ефективного вирішення NP-повних задач, була створена нова наука – «Теорія алгоритмів». Алгоритми можуть бути чисельними й логічними.

Алгоритми, що призначені для розв’язування задачі у вигляді арифметичних дій над числами, називають чисельними алгоритмами [1].

Логічні алгоритми, як правило, полягають у маніпулюванні даними з метою встановлення або визначення деякої закономірності. До логічних алгоритмів можна віднести алгоритми класифікації, сортування, планування та ін [1]. Досвід розробки й застосування алгоритмів дозволяє викоремити ряд загальних властивостей будь-яких алгоритмів [1]:

- **дискретність алгоритму**. Ця властивість визначає, що будь-який алгоритм можна розглядати як процес послідовного виконання певних команд, що йде у дискретному часі, у ході виконання певного набору інструкцій;

- **масовість алгоритму**. Ця властивість визначає, що алгоритм має розв’язувати не тільки конкретну визначену задачу, але може застосовуватися для розв’язання цілого класу однотипних або подібних задач;

- **детермінованість алгоритму**. Ця властивість визначає, що після виконання чергового етапу чітко визначено, що робити на наступному етапі;

- **елементарність кроків алгоритму**. Ця властивість визначає, що розв’язування задачі розбивають на етапи, кожний з яких повинен бути простим і локальним, тобто відповідна операція має бути елементарною для

виконавця алгоритму (людини або машини);

- **результативність алгоритму.** Ця властивість визначає, що алгоритм через скінченну кількість кроків має привести до досягнення необхідного результату.

Найбільш наочним способом представлення алгоритму є блок-схема. Блок-схема це наочне графічне зображення алгоритму, у якому окремі етапи зображують за допомогою різних геометричних фігур – блоків.

Конфігурацію і розмір блоків, а також порядок графічного оформлення блок-схем регламентує ДЕРЖСТАНДАРТ 10.002-80 і ДЕРЖСТАНДАРТ 10.003-80 «Схеми алгоритмів і програм» [1]. У більшості випадків спосіб відеозображення відповідає міжнародним стандартам. Блоки супроводжують написами.

У таблиці 2 наведено блоки, зображені елементи зв’язків між ними, а також надано коротке пояснення до них.

Таблиці 2 цілком достатньо для зображення алгоритмів, які необхідні при виконанні студентських лабораторних робіт, які мають бути виконані в ході вивчення дисциплни.

У таблиці 3 показані з'єднувачі і додаткові елементи блок-схем алгоритмів. При з’єднанні блоків слід використовувати тільки вертикальні й горизонтальні лінії.

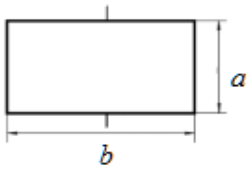
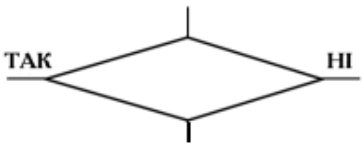
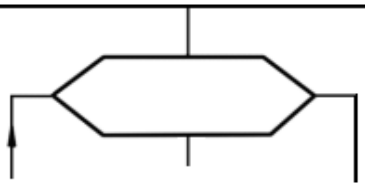
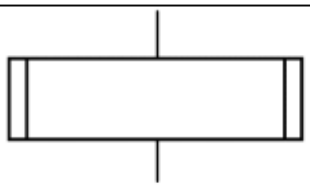
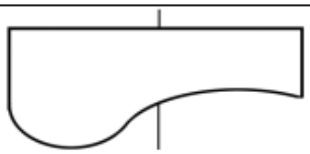
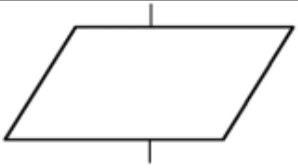
Відстань між паралельними лініями потоків повинна бути не менше 3 мм, між іншими елементами схеми – не менше 5 мм. Горизонтальний і вертикальний розміри блоку мають бути кратні 5 (ділитися на 5 націло).

Співвідношення горизонтального й вертикального розмірів блоку b/a = 1.5 є основним. При ручному виконанні креслення блоку припустиме співвідношення b/a = 2. В ході підготовки звітів лабораторних робіт необхідно притримуватися вказаних вимог до оформлення блок-схем алгоритмів, які реалізуються в учбовому завданні.

Правильно сформована блок-схема алгоритма дозволяє спростити

написання коду програму, що реалізує відповідний алгоритм.

Таблиця 2 — Основні елементи блок-схем алгоритмів [1]

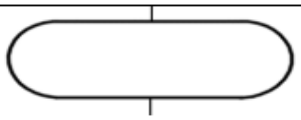
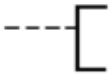
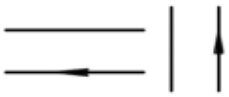
Назва	Елемент	Коментар
Процес		Обчислювальна дія або послідовність обчислювальних дій
Розв'язування		Перевірка умови
Модифікація		Заголовок циклу
Визначений процес		Звертання до процедури
Документ		Вивід даних, друк даних
Введення/Виведення		Ввід/Вивід даних

Блоки «Початок», «Кінець» і «З'єднувач» мають висоту *a*/2, тобто вдвічі меншу за основну висоту блоків. Для розміщення блоків рекомендується поле аркуша розбивати на горизонтальні і вертикальні (для схем, що розгалужуються) зони. Для зручності опису блок-схеми кожний її блок слід пронумерувати, використовувати наскрізну нумерацію блоків. Номер блоку розташовують у розриві в лівій верхній частині рамки блоку.

По характеру зв'язків між блоками розрізняють алгоритми лінійної,

циклічної структури, та структури, що розгалужується.

Таблиця 3 — Додаткові елементи блок-схем алгоритмів [1]

Назва	Елемент	Коментар
З'єднувач		Розрив лінії потоку
Початок, Кінець		Початок, кінець, пуск, зупинка, вхід і вихід у допоміжних алгоритмах
Коментар		Використовують для розміщення написів
Горизонтальні й вертикальні потоки		Лінії зв'язків між блоками, напрямки потоків
Злиття		Злиття ліній потоків
Міжсторінковый з'єднувач		Перехід на інший лист

Лінійний алгоритм – це алгоритм, у якому блоки виконуються послідовно зверху вниз від початку до кінця. Це самий простий приклад алгоритму. У першій частині даної лабораторної роботи реалізований саме лінійний алгоритм. У реальному програмуванні лінійні алгоритми практично не зустрічаються. Звичайно процес виконання певної програми залежить від певних умов і проходить по різним гілкам алгоритму. Такий алгоритм називають алгоритмом, що розгалужується. Для зображення процесу розгалуження у блок-схемах використовують блок «Розв'язування», що показаний у табл.2. За допомогою блоку «Розв'язування» в алгоритмі виконується перевірка умови. Важливо формувати умову у цьому блоці таким чином, щоб однозначно давати відповіді чи умова виконується, чи умова не виконується. Кількість гілок залежить від умов, що перевіряються, при наявності декількох гілок використовуються декілька блоків «Розв'язування».

На теперішній час у Інтернеті є багато хибних прикладів алгоритмів. Отже треба бути обережним при використанні цих прикладів як шаблонів і намагатися дотримуватися класичних правил побудови і зображення алгоритмів. Для реалізації блоку «Розв'язування» у мові С є певний набір операторів. Це оператори вибору або оператори управління ходом виконання програми. Вони дають можливість побудувати програму, що дозволяє створити вибір однієї з альтернативних гілок програми, перевіряючи для цього певні умови. Такими операторами є: *if, if...else* та *switch*. Базовий оператор *if...else* має наступний шаблон:

```
if (вираз)
{
        блок операторів, якщо вираз повертає true
}
else
{
        блок операторів, якщо вираз повертає false
}
```

Наведемо практичний приклад використання даного оператору на прикладі завдання до лабораторної роботи. У коді, що наведено далі, показаний тільки практичний приклад використання оператора *if...else* для одного з варіантів реалізації алгоритму розгалуження. Тобто цей код призначений виключно для демонстрації.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main()
{
   // оголошення змінних, які будуть використовуватися в програмі
   unsigned int variant; // 1 або 2  — допустимі значення змінної variant
   int N;
   int n;
```

```
    double start, delta, x;

    // виконуємо введення змінної variant
    // При введені недопустимих значень відбувається повторне введення
    // за допомогою циклу while()
    printf("Enter variant (1 or 2 ): ");
    scanf("%u", &variant);

    if (variant==1) {
      printf("Perform Action for var 1\n");
    }
    else  if (variant==2) {
      printf("Perform Action for var 2\n");
    }  else
       printf("Variant not valid \n");
printf("Program finished\n");
return 0;
}
```

Блок-схема алгоритму, що відповідає даному вихідному коду наведена далі на рис. 7. У відповідності до таблиці 2 у блок-схемі (БС) алгоритму у блоці 3 відбувається виведення запрошення на введення варіанту виконання. Далі у блоці 5 відбувається перевірка чи містить змінна *variant* значення 1. Якщо так, то програма починає виконуватися по крайній лівій гілці БС. Умовно виконується дія варіанту 1 і далі відбувається вивід повідомлення у блоці 8 про завершення програми. Аналогічно працює алгоритм у тому варіанті коли змінна містить значення 2. Для всіх інших значення номеру варіанту введення початкових даних виводиться повідомлення про хибний номер варіанту і програма завершує свою роботу.

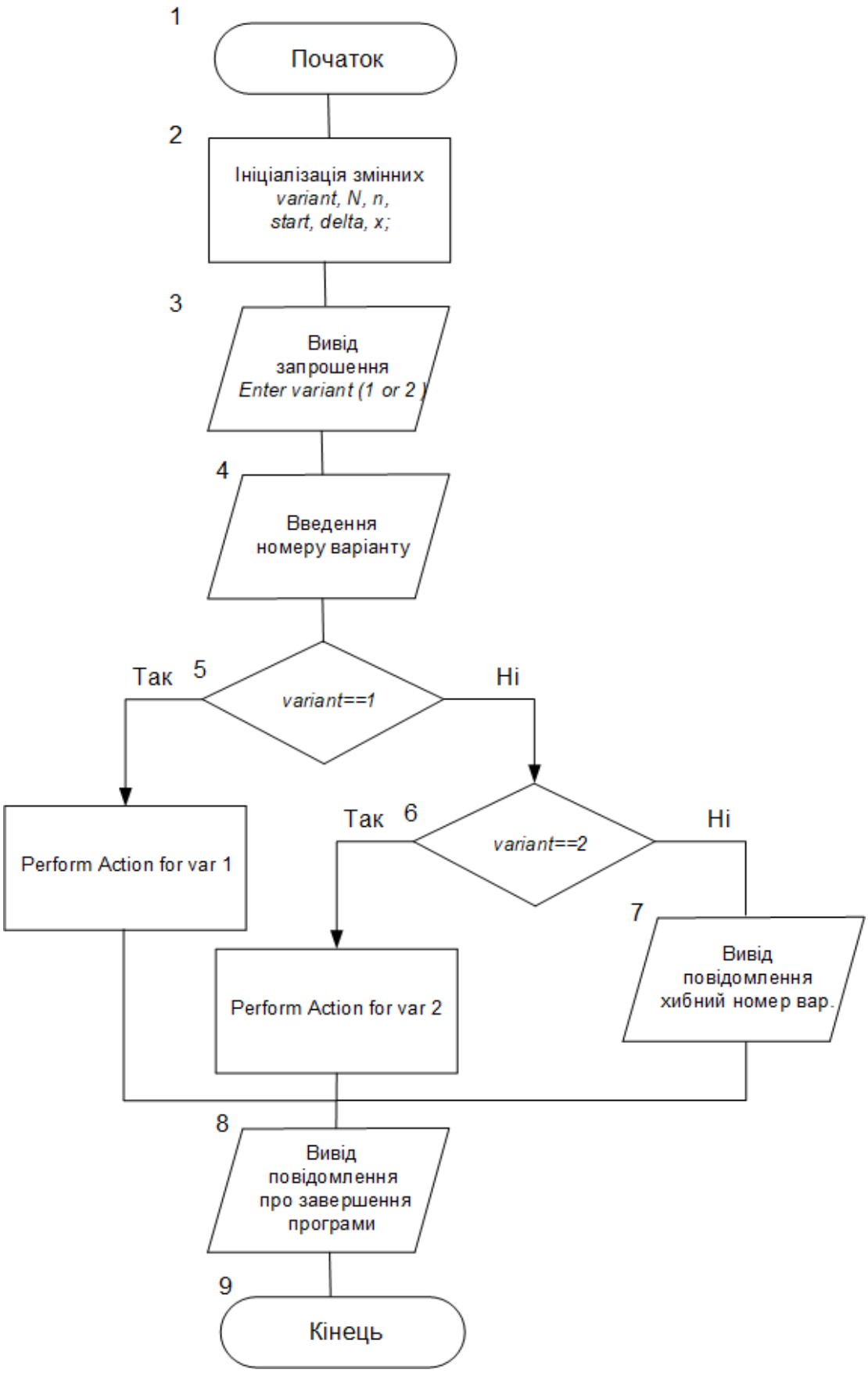


Рис. 7 — Приклад алгоритму розгалуження

Показане рішення демонструє роботу алгоритму розгалуження. Недоліком такого рішення є те, що для виконання умов лабораторної роботи існує необхідність кожний раз запускати програму при хибному значенні варіанту. Наведемо практичний приклад алгоритму для ЛР 1, що більш придатний для вибору правильного варіанту. Перед розглядом прикладу треба зауважити, що відповідно до завдання можуть бути тільки два варіанта рішення. *Варіант* повинен дорівнювати 1 або 2. У всіх інших випадках програма має знов виводити запрошення введення варіанту і відпрацьовувати алгоритм перевірки номеру варіанту ще раз, поки не буде введено значення 1 або 2 для варіанту введення початкових даних.

Розглянемо приклад вихідного коду, що буде корисним для вибору і валідації номеру варіанту введення початкових даних.

Якщо номер варіанту дорівнює або 1 або 2, то програма друкує номер варіанту в консоль, у іншому випадку запропоновано ввести номер варіанту ще раз. У цьому прикладі використовується циклічна конструкція. При розв’язуванні задач доводиться повторювати виконання певного набору операцій і зробити багаторазовий прохід по певним ділянкам алгоритму. Такі ділянки називають тілом циклу. Алгоритми, що містять цикли, називають циклічними. Використання циклів суттєво скорочує обсяг алгоритму. У нашому прикладі це робить вихідний код більш читабельним. Типові етапи організації циклу наступні:

- підготовка (ініціалізація) циклу;
- виконання обчислень або корисного функціоналу циклу (тіло циклу);
- модифікація параметрів, що впливають на умови виконання циклу;
- перевірка умови закінчення циклу і виконання відповідної дії — повторення циклу або вихід з циклу.

Розрізняють цикли з апріорно відомими або апріорно невідомими кількостями проходів через тіло циклу.

Цикл називають ітераційним, якщо число повторень тіла циклу

заздалегідь невідомо, залежить від значень параметрів (деяких змінних), що беруть участь у обчисленнях. Приклад канонічного варіанту БС ітераційного варіанту циклу показаний на рис. 8 [2].

У мові С або інших С-подібних мовах програмування оператори ітерації дозволяють організовувати циклічне виконання набору операторів програми.

Існує три форми операторів ітерації:

- *while*
- *do...while*
- *for*

Оператор *while*, що застосований у прикладі вихідного коду має наступний загальний формат:

```
while (умовний вираз)
{
        блок операторів поки результат умовного виразу є ІСТИНА;
}
```

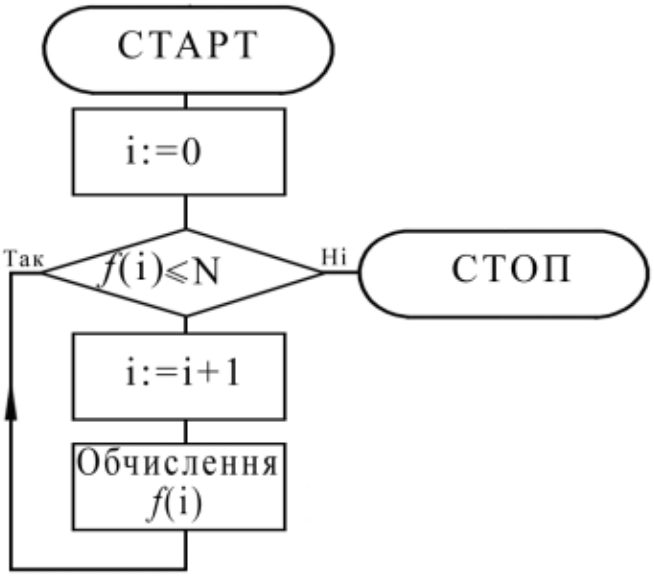


Рис. 8 — Алгоритм з ітераційним циклом [1]

Саме такий варіант запропонований для рішення задачі вибору правильного номеру варіанту.

Умовний вираз для *while* обчислюється і перевіряється першим. Якщо це

значення ненульове (ІСТИНА), то виконується блок операторів що належить до нього. Цикл повторюється, поки умовний вираз не дасть значення 0.

У блоці операторів має бути механізм, що дозволяє певним чином впливати на значення умовного виразу *while* з метою запобігання зациклення програми при випадку, коли результат логічного виразу буде завжди ІСТИНА.

Наведемо узагальнений приклад БС алгоритму для рішення завдання другої частини ЛР № 1.

Блок-схема зображена на рис. 9. У алгоритмі у блоці 2 відбувається оголошення і ініціалізація змінних.

Далі відбувається виведення в консоль або термінал запрошення користувача програми на введення варіанту виконання (блок 3). Далі у блоці 4 БС відбувається введення варіанту користувачем програми. Перевірка чи містить змінна *variant* значення 1 або 2 здійснюються за рахунок ітераційного циклу. Отже для рішення задачі вибору варіанту використаний саме ітераційний варіант циклу. У блоці 5 відбувається перевірка відповідності варіанту 1 або 2. Якщо варіант відповідає допустимому значенню, то програма продовжує роботу. У іншому випадку програма буде виконувати тіло ітераційного циклу нескінченне число разів, поки не буде введено номер варіанту 1 або 2, тобто будуть виконуватися блоки 3 і 4 алгоритму. Існують інші варіанту циклічних конструкцій, у яких число повторень апріорно відомо. Такий цикл називають детермінованим. Узагальнений приклад такого циклу показаний на БС на рис. 10.

Оволодіння циклічними операторами є вкрай важливим при реалізації різноманітних алгоритмів. При цьому потрібно бути дуже прискіпливим до формування логічних виразів, які перевіряються, і від результату яких цикл буде продовжуватися чи закінчуватися. Одним із проблемних моментів може бути зациклення програми у випадку, коли, наприклад, логічний вираз буде завжди істинний і ніяких інших механізмів виходу із циклу не передбачено. Це може стати серйозною логічною помилкою.

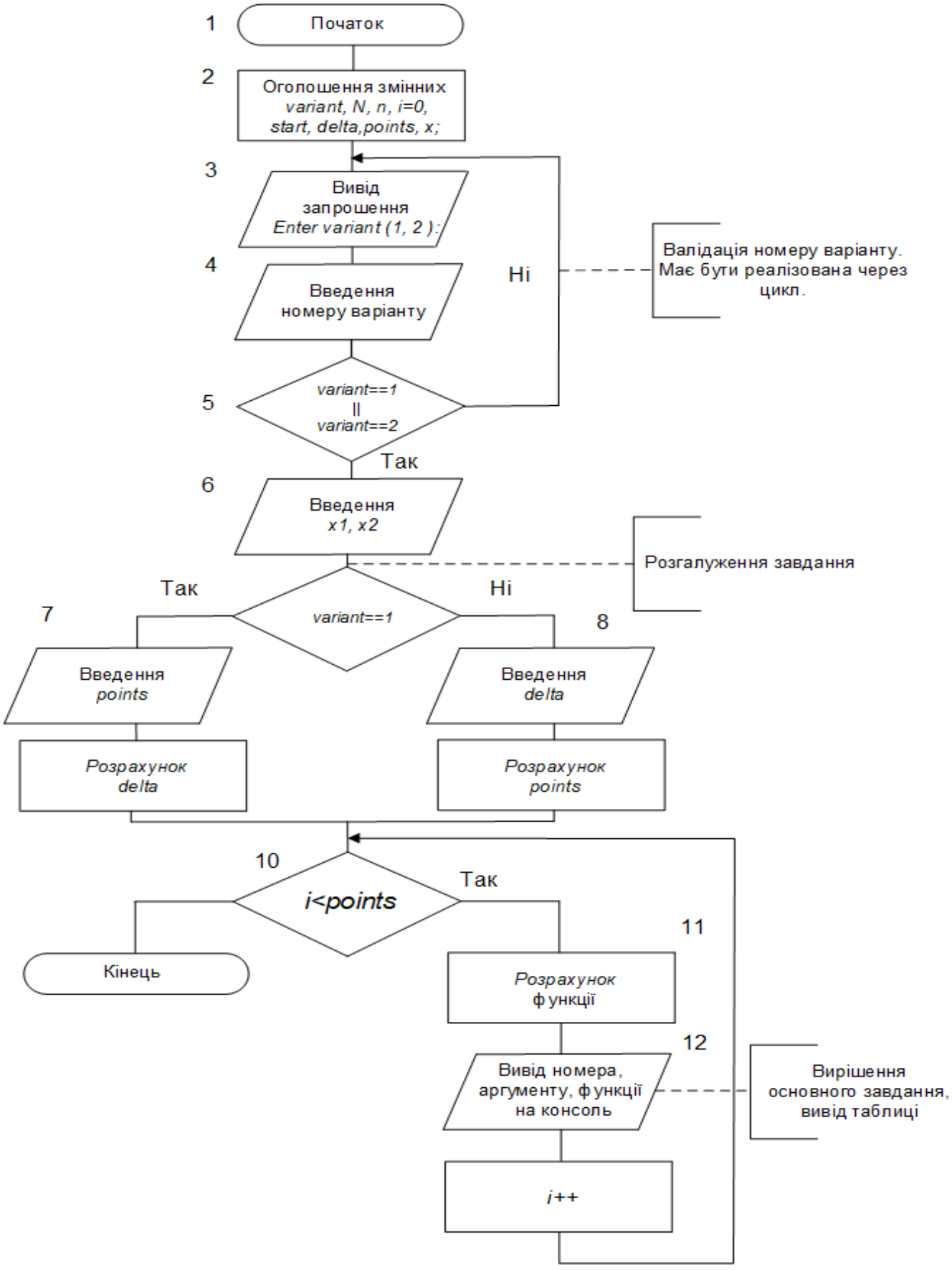


Рис. 9 — Блок-схема рішення

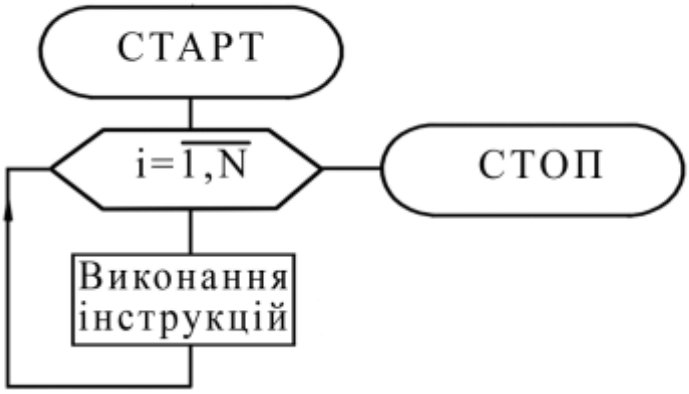


Рис. 10 — Алгоритм з детермінованим циклом [1]

Цей приклад демонструє, що інструкція буде виконуватися рівно N разів. Крім описаного вище оператору для організації циклу у мові С використовуються інші оператори.

Оператор *do...while* має наступний загальний формат:

```
do
{
    блок операторів поки результат умовного виразу є ІСТИНА;
} while (умовний вираз);
```

На відміну від оператору *while* у операторі *do...while* спочатку виконується блок операторів, після цього обчислюється значення умовного виразу. Якщо це значення ненульове (ІСТИНА), то виконується блок операторів, що знаходиться вище.

Цикл повторюється, поки умовний вираз не дасть значення 0 (НЕПРАВДА, ХИБА). У блоці операторів має бути також механізм, що дозволяє певним чином впливати на значення умовного виразу в циклічному операторі *do...while* з метою запобігання зациклення програми.

Також для організації циклу у мові С є оператор *for*, що забезпечує відносно компактний варіант організації циклів.

Формат оператора for наступний:

for(<Оператор 1. Зазвичай це ініціалізація змінних> ; <Оператор 2. Зазвичай умовний вираз> ; <Оператор 3. Зазвичай інкремент управляючої змінної>)

{

блок операторів, поки оператор 2 в результаті виконання дає ІСТИНУ;

}

Послідовність дій при виконанні циклічного оператора *for*:

– однократно виконується оператор 1, зазвичай це ініціалізація змінної, яка використовується в логічному виразі, від результату якого залежить продовження циклу;

– виконується оператор 2, зазвичай це обчислення умовного виразу. Якщо отримано ненульове значення (ІСТИНА), виконується блок операторів, якщо умовний вираз дає значення 0 (НЕПРАВДА, ХИБА), виконання циклу припиняється;

– виконується оператор 3, звичайно це операція інкременту (декременту) або інший який-небудь варіант модифікації змінної (лічильника), яка використовується в операторі 2 (в логічному виразі). Після оператору 3 управління передається на оператор 2, де зазвичай виконується обчислення логічного виразу і в залежності від його результату відбувається вихід із циклу або перехід до наступної ітерації циклу.

Таким чином, в якості початкового шаблону БС алгоритму лабораторної роботи № 1 можна використовувати БС рис. 9, або рис. 10 та рис. 11.

Далі потрібно побудувати таблицю функції $f(x)$ відповідно до завдання і варіантів наведених у табл.1. Це дозволить визначити проміжки $[\alpha, \beta]$ на яких є рішення рівняння. Приблизний розв'язок визначається як точка перетинання функції $f(x)$ та осі x (абсцис), а у таблиці це буде відрізок на якому відбувається зміна знаків значення $f(x)$.

У таблиці 4 наведені варіанти для побудови текстового інтерфейсу виводу інформації на екран.

Таблиця 4 — Варіанти створення таблиці

Варіант	Ширина стовпця (символів)			Варіант	Ширина стовпця (символів)		
	"номер точки"	"аргумент"	"значення функції"		"номер точки"	"аргумент"	"значення функції"
1	5	20	25	2	8	23	23
3	7	24	29	4	12	26	27
5	5	18	21	6	6	25	25
7	10	27	24	8	6	29	26
9	3	22	28	10	7	21	21
11	4	21	21	12	9	24	24
13	7	29	19	14	4	28	28
15	5	26	23	16	11	22	26
17	6	24	24	18	5	21	25
19	11	28	28	20	9	23	29
21	7	21	22	22	4	27	22
23	5	19	20	24	9	25	23
25	9	23	22	26	8	26	27
27	8	24	24	28	7	25	25
29	4	24	25	30	5	26	26

2.3. Контрольні запитання до лабораторної роботи № 1

1. Яка структура простої програми на мові С?
2. Наведіть приклад запису функції *main()* для простої програми?
3. Вкажіть етапи обробки коду для отримання файлу із розширенням .exe?
4. Що таке ідентифікатор? Що таке службові слова? Наведіть приклади службових слів. Чи можна використовувати службові слова в якості ідентифікаторів?
5. Назвіть правила складання ідентифікаторів? Наведіть приклади правильно створених ідентифікаторів?
6. Яким чином оголошуються константи в мові програмування С? Яка різниця між іменованими та неіменованими константами?
7. Назвіть базові типи даних: розмір в байтах, діапазон значень?
8. Який синтаксис оголошення та ініціалізації змінних?
9. Особливості застосування префіксної та суфіксної форм операції інкременту та декременту у виразах?
10. Що таке специфікатор формату?

3. ЛАБОРАТОРНА РОБОТА № 2 «ОБЧИСЛЕННЯ ВИЗНАЧЕНОГО ІНТЕГРАЛУ»

Мета роботи: полягає у вдосконаленні знань, умінь та навичок з технології розроблення програмного забезпечення (ПЗ) з використанням мови С у процедурній парадигмі. Надається досвід створення блок-схем алгоритмів.

Також лабораторна робота дає основні навички обчислення визначеного інтегралу чисельними методами. Дається можливість роботи з іншим типом IDE за вибором студента та по узгодженню з викладачем. Лабораторна робота складається з двох частин. Підготовлені студенти можуть відразу виконувати другу (основну) частину лабораторної роботи без зменшення максимальної кількості балів, що зараховуються за повне і правильне виконання завдання.

Перша частина лабораторної роботи передбачає розробку програми і виконання завдання для вивчення операторів розгалуження мови програмування С і підготовки до другої частини лабораторної роботи. Рівень складності лабораторної роботи першого етапу відповідає рівню складності лабораторної роботи № 2, 3 описаної у [2].

Друга частина лабораторної роботи передбачає розробку програми чисельного розрахунку визначеного інтегралу для закріплення знань, що отримані на курсі вищої математики і підготовки до наступних лабораторних робіт.

3.1. Частина перша. Реалізація оператору розгалуження

Вхідні дані ЛР2. Частина 1

Функція, реалізація якої залежить від значення аргументу, перелік функцій показаний у розділі *Завдання,* функцію обрати відповідно до варіантів.

Математичну функцію відповідно до варіанту завдання необхідно описати як власну функцію, яку розробник має запрограмувати на мові програмування С.

Вихідні дані ЛР2. Частина 1

Проект IDE Code::Blocks, вихідний код програми мовою С, що виводить на консоль (термінал) значення функції, яка залежить від значення аргументу. Розраховане значення функції, яке виводиться на екран монітора.

Завдання

1. У якості індивідуального завдання написати код, що виводить на екран консолі (терміналу) значення функції, що залежить від аргументу.

2. Запрограмувати вираз знаходження значення *y* в залежності від введеного значення *x* відповідно до свого варіанту. У лабораторній роботі необхідно реалізувати програму, що виконує розрахунки функції в залежності від введеного значення *x*. Наприклад, розрахувати значення змінної *y*:

$$y = \begin{cases} \dfrac{1}{1+\cos(x)}, & 0 \le x < 1 \\ e^{x+2}, & 1 \le x < 2 \\ 2^x + 2x - 4, & 4 \le x < 8 \\ 0, & \textit{для усіх інших значень } x \end{cases}$$

Передбачається, що змінна *x*, а також результат обрахунку, що зебрігається в змінній *y*, мають тип даних *double*. Варіанти виконання першої частини лабораторної роботи представлені нижче.

3. Вихідний програмний код розмістити на сайті https://github.com/. Розмістити або у відкритому або у приватному варіанті із наданням прав доступу викладачу для первинної перевірки.

4. Звіт до першої частини лабораторної роботи оформити відповідно до вказаних вимог. Звіт також розташувати на https://github.com/.

Варіанти завдання

Варіант № 1.

$$y=\begin{cases} x^2+2x+1, & -5\le x<0 \\ \sqrt{x+1}-\dfrac{1}{\sqrt{x+5}}, & 0\le x<10 \\ \log_{10}(x)+2x, & 10\le x<1000 \\ \dfrac{x}{2}, & \textit{для усіх інших значень } x \end{cases}$$

Варіант № 2.

$$y=\begin{cases} \dfrac{x-1}{x+1}, & 0\le x<5 \\ 5x+10, & 5\le x<25 \\ \dfrac{x}{\sqrt{x+x^2}}-4x^3, & 100\le x<125 \\ \dfrac{x}{10}+4, & \textit{для усіх інших значень } x \end{cases}$$

Варіант № 3

$$y=\begin{cases} 10\sin(x)-5\cos(2x), & 0\le x<3 \\ x^2+2x-16, & 5\le x<8 \\ \log_{10}(x^2+x)-3x, & 20\le x<1000 \\ -4, & \textit{для усіх інших значень } x \end{cases}$$

Варіант № 4

$$y=\begin{cases} x^3+2x^2, & -15\le x<-5 \\ 5, & 2\le x<18 \\ \sqrt{x^2+10x}, & 58\le x<104 \\ -2-\dfrac{x}{25}, & \textit{для усіх інших значень } x \end{cases}$$

Варіант № 5

$$y=\begin{cases}2^{|x|}, & -20\le x<-10\\ \dfrac{|\sin(x)|}{\cos(x)+2}, & -2\le x<2\\ 2x+\sqrt{x^2-4}, & 5\le x<10\\ \dfrac{x}{5}+\dfrac{x-1}{2}, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 6

$$y=\begin{cases}\ln\left(\dfrac{x^2}{10}\right)+2, & 10\le x<100\\ \log_{10}(x)+\dfrac{x^2-4}{x^3+5}, & 100\le x<1001\\ -3\sin^2(x)+3, & 1800\le x<2000\\ 1, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 7

$$y=\begin{cases}|x+4|-|x^3-4|, & -3\le x<0\\ 2x, & 9\le x<15\\ \dfrac{x}{\cos^2(x)+1}, & 27\le x<50\\ \dfrac{x}{4}+\dfrac{\sin(x)}{3}, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 8

$$y=\begin{cases}-x, & -100\le x<-50\\ x, & 50\le x<100\\ \log_{10}(x)+1, & 200\le x<1000\\ 0, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 9

$$y=\begin{cases}\dfrac{2}{x}-1, & 0.5\le x<1.5\\ x-2, & 1.5\le x<5.125\\ \dfrac{\sqrt{(x+1)}}{\sqrt{x^2-10}}-x^3, & 5.125\le x<9.5\\ 1, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 10

$$y=\begin{cases}e^{-x}+1, & -5.2\le x<0\\ e^{x}-1, & 0\le x<8\\ \dfrac{\sqrt{(2x-7)}}{\sqrt{x^2-25}}+2x^{e}, & 8\le x<15\\ 0, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 11

$$y=\begin{cases}2x^2-e^{-x}+1, & -5\le x<1\\ \sqrt{2x+1}-\dfrac{4}{\sqrt{4x+1}}, & 1\le x<100\\ 2\log_{10}(2x)-\dfrac{x^2}{4}, & 100\le x\le 1000\\ -1, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 12

$$y=\begin{cases}\dfrac{2x-1}{3x+1}, & 0\le x<4\\ 5x+10, & 4\le x<12\\ \dfrac{2x}{\sqrt{-x+2x^2}}-3x^2, & 15\le x<55\\ 2-\dfrac{x}{10}, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 13

$$y=\begin{cases}4\sin(3x)+3\cos(4x), & -3\le x<3\\ 2x^2-5x+9, & 10\le x<20\\ \dfrac{\log_{10}\left(x^2\right)}{2}-3x, & 100\le x<1000\\ -\dfrac{x}{10}+1, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 14

$$y=\begin{cases}2x^3-4x^2, & -20\le x<-1\\ 5, & -1\le x<1\\ \sqrt{2x^2-5x}, & 5\le x<15\\ 1-\dfrac{x}{2}, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 15

$$y=\begin{cases}2^{|x|+1}, & -12\le x<-8\\ \dfrac{|\cos(x)|+x}{\sin(x)+1}, & -3\le x<3\\ 4x+\sqrt{x^3-2x}, & 10\le x<200\\ \dfrac{1}{125}, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 16

$$y=\begin{cases}4\ln\left(\dfrac{x^2}{2}\right)-\sqrt{x}, & 3\le x<10\\ \log_{10}\left(\dfrac{x}{2}\right)+\dfrac{\left|4-x^2\right|}{2x-1}, & 200\le x<2000\\ -4\sin\left(\dfrac{x}{2}\right)-\cos(x+1), & 2000\le x<2500\\ -0.5, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 17

$$y=\begin{cases}\dfrac{|x-3|}{|x^3-5|}-e^{x+1}, & -4\le x<-1\\ 3x+1, & 2\le x<5\\ \dfrac{1}{x}, & 10\le x\le 20\\ x-1, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 18

$$y=\begin{cases}e^{x-1}, & -5\le x<-2\\ e^{x+1}, & 2\le x<5\\ -\log_{10}(x)+\dfrac{x}{\cos(2x)+1}, & 10\le x<15\\ 5, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 19

$$y=\begin{cases}\dfrac{\cos(x)}{2}, & -2\le x<2\\ \dfrac{x^2}{e^x}+\sqrt{x-1}, & 4\le x<8\\ \dfrac{\ln(x+1)}{x+1}, & 10\le x\le 20\\ \dfrac{x}{1000}-4, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 20

$$y=\begin{cases}\sqrt{|x+2|+\dfrac{|\cos(x)|}{2}}, & -10.5\le x<8.125\\ 4, & 10\le x<15\\ 2^{\log_{10}(100)}-4x, & 50\le x<150\\ -1.5, & \textit{для усіх інших значень } x\end{cases}$$

Варіант № 21

$$y=\begin{cases} x^3-2x^2+|x|, & -10\le x<-2 \\ 2\sqrt{x-2}-\left(\dfrac{x^3-1}{\sqrt{x+2}}\right)^3, & 4\le x<12 \\ \dfrac{\log_{10}(2x)}{4}, & 100\le x<10000 \\ -\dfrac{x}{5}+2, & \textit{для усіх інших значень } x \end{cases}$$

Варіант № 22

$$y=\begin{cases} \dfrac{2x-1}{2x+1}, & 0\le x<2 \\ -x+1, & 2\le x<8 \\ \dfrac{2x}{\sqrt{x^3+1}}-5x^2-\cos(x), & 20\le x<50 \\ 0, & \textit{для усіх інших значень } x \end{cases}$$

Варіант № 23

$$y=\begin{cases} \dfrac{1}{2}\cos\left(\dfrac{x}{2}\right)-\dfrac{1}{x^2}\sin(2x), & 0\le x<3 \\ 4-x^2, & 7\le x<10 \\ \dfrac{10}{x}+x^2, & 10\le x<30 \\ 2, & \textit{для усіх інших значень } x \end{cases}$$

Варіант № 24

$$y=\begin{cases} -x^3+1, & -10\le x<-2 \\ \dfrac{1}{2x}, & 3\le x<8 \\ \dfrac{x^3}{\sqrt{x-1}}, & 12\le x<20 \\ 0, & \textit{для усіх інших значень } x \end{cases}$$

Варіант № 25

$$y=\begin{cases} 3^{|x-1|}-\dfrac{|x+1|}{e^2}, & -10\le x<-5 \\ \dfrac{2\left|\sin\left(\dfrac{x}{2}\right)\right|}{\cos(x-1)+5}, & -1\le x<1 \\ 0.5, & 5\le x<10 \\ \dfrac{x}{100} & \textit{для усіх інших значень } x \end{cases}$$

Програма проведення експерименту ЛР 2. Частина 1

Створити проект і вихідний код, налагодити програму, виконуючи наступну послідовність дій: ,

- запустити програму, яка наведена у розділі "*Теоретичні відомості ЛР 2. Частина 1*" та перевірити правильність виконання;
- провести первинний аналіз функції, що надана у завданні, розробити формалізований опис задачі, обрати узагальнену модель рішення;
- розробити алгоритм рішення завдання, алгоритм можна розробити у вигляді вихідного коду або блок-схеми алгоритму;
- провести проектування структури програми. Для даного завдання краще вибрати структуру у вигляді одного файлу;
- створити проект у IDE Code::Blocks, зробити зміни у вихідному коді шляхом використання текстового редактора відповідно до завдання;
- зробити компіляцію і переконатися у правильній роботі програми, виправити синтаксичні помилки у випадку їх наявності;
- провести тестування програми, порівняти результати роботи програми і результати обчислень функції з використанням калькулятора або спеціалізованих програмних пакетів для інженерних обрахунків та математичних обчислень. Якщо результати однакові, тоді можна зробити висновки щодо правильності роботи програми.

Теоретичні відомості ЛР 2. Частина 1

Розглянемо більш докладно фундаментальні типи даних, що використовуються у мові програмування С і коротко описані у ЛР № 1. Для таких типів даних є інші назви, наприклад вбудовані типи даних, іноді ще використовується термін «примітиви». До фундаментальних типів належать наступні: *void, char, int, float, double* разом з модифікаторами *short, long, signed, unsigned*. Похідні типи даних включають вказівники та посилання на інші типи, масиви, структури і об'єднання. У мові програмування С або С++ ці типи пов'язані з архітектурою мікропроцесора або мікроконтролера [4]. Виконання лабораторної роботи передбачає використання математичних функцій мови програмування С, а також дає можливість засвоїти принципи роботи умовних операторів вибору if та if/else. В мові програмування С передбачені такі арифметичні операції:

*	множення
/	ділення
%	остача від ділення (застосовується до цілих типів)
+	додавання
–	віднімання

Крім арифметичних операцій, також широко використовуються операції відношення, які застосовуються при формуванні логічних виразів в умовних операторах. Нижче наведені операції відношення:

<	менше
<=	менше або рівне
>	більше
>=	більше або рівне
==	рівне

!= не рівне

Для формування логічних виразів використовуються логічні операції:

! логічне НЕ

&& Логічне І

|| Логічне АБО

Для використання математичних функцій, необхідно в текст програми помістити директиву препроцесора, яка приєднує заголовочний файл *math.h*:

#include <math.h>

Нижче у табл.5 наведені окремі математичні функції, що належать до бібліотеки *math.h*.

Таблиця 5 — Математичні функції мови С

Назва функції	***Опис***	***Прототип функції***
abs()	Повертає модуль цілого числа. Приклад виклику функції: *y* = abs(*x*); *y* та *x* - змінні типу int	int abs(int arg)
fabs()	Повертає модуль дійсного числа. Приклад виклику функції: *z* = fabs(*a*); *z* та *a* - змінні типу double або float	double fabs (double arg)
exp()	Повертає значення $e^{\arg}$. Приклад виклику функції: *z* = exp(*a*); Змінній *z* присвоюється значення, яке обраховується як e^{a}	double exp (double arg)

Таблиця 5 — Математичні функції мови С (продовження)

Назва функції	***Опис***	***Прототип функції***
log10()	Повертає значення десяткового логарифму аргумента arg. Приклад виклику функції: z = log10(a); Змінній z присвоюється значення яке обраховується як $\log_{10}(a)$	double log10 (double arg)
pow()	Повертає значення, яке розраховується шляхом піднесення значення аргументу b до степіні a. Приклад виклику функції: *z* = pow(*b*,*a*); Змінній *z* присвоюється значення яке обраховується як b^a	double pow (double b, double a)
sin()	Повертає синус аргументу arg. Приклад виклику функції: *z* = sin(*a*) Змінній *z* присвоюється значення яке обраховується як $\sin(a)$	double sin (double arg)
cos()	Повертає косинус аргументу arg. Приклад виклику функції: *z* = cos(*a*) Змінній *z* присвоюється значення яке обраховується як $\cos(a)$	double cos (double arg)
tan()	Повертає тангенс аргументу arg. Приклад виклику функції: *z* = tan(*a*) Змінній *z* присвоюється значення яке обраховується як $\mathrm{tg}(a)$	double tan (double arg)

Таблиця 5 — Математичні функції мови С (продовження)

Назва функції	***Опис***	***Прототип функції***
sqrt()	Повертає квадратний корінь аргумента arg. Приклад виклику функції: z = sqrt(a) Змінній z присвоюється значення яке обраховується як $\sqrt{a}$	double sqrt (double arg)
log()	Повертає значення натурального логарифму аргумента arg. Приклад виклику функції: z = log(a); Змінній z присвоюється значення яке обраховується як ln(a)	double log (double arg)

Приклад тексту програми.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
int main()
{
   double x, y;
   printf("Enter x:");
   scanf("%lf", &x);
    if( x>= 0  &&  x < 1)
      y = 1.0 / (1.0 + cos(x));
   else if ( x >= 1  &&  x < 2 )
      y = exp(x+2);
   else if ( x >= 4  &&  x < 8 )
      y = pow(2,x)+2*x-4;
   else
      y = 0.0;
```

```
    system("cls");
    printf("x = %lf", x);
     printf("\ny = %lf", y);
    return 0;
}
```

Звичайно, кожна змінна має бути оголошена перед початком її використання і ініціалізована. Значення ініціалізації для змінної може представляти собою будь-який вираз, що дає при обчисленні значення, сумісне з типом змінної, що буде присвоєне змінні, або ж змінній може бути присвоєно значення відповідної іменованої або неіменованої константи. При невідповідності типів даних змінної та результату, що їй присвоюється, відбувається автоматичне перетворення типів даних. В разі, коли у виразі використовуються змінні різних типів даних може бути необхідним виконувати явне перетворення типів даних, застосовуючи для цього відповідну операцію мови С.

Розглянемо більш докладно оператори розгалуження, що були описані у ЛР № 1. Операторів розгалуження у мові С, С++, або у С-подібних мовах програмування (Java, C#, тощо) три. Це оператори *if*, *if/else* та *switch*. Перші два опепатори дозволяють зробити дві гілки у програмі, у цілому здійснити розгалуження по одному з двох напрямків. Другий дозволяє зробити вибір між великим числом варіантів у компактному виразі. Робота оператора *if* та *if/else* описана у ЛР № 1. Розглянемо оператор switch.

Оператор switch забезпечує лаконічний спосіб перемикання між різними частинами програмного коду в залежності від значення змінної або виразу.

Загальна форма цього оператора така:

```
switch (вираз, що повертає ціле значення або змінна)
{
    case значення_1:
                break;
```

```
        case значення_2:
                break;
        case значення_N:
                break;
        default:
    }
```

Обчислене значення виразу, що обов'язково має ціле значення, порівнюється з усіма значеннями, зазначеними у операторах *case*. Якщо вираз збігається зі значенням біля *case*, управління передається коду, що знаходиться за ним (після двокрапки). Якщо ж значення виразу не відповідає жодному з операторів case, управління передається коду, розташованому після службового слова *default*. Відзначимо, що оператор *default* необов’язковий. У випадку, коли жоден з операторів *case* не відповідає значенню виразу, і в *switch* відсутній оператор *default*, виконання програми продовжується з оператора, що є наступним після оператора *switch*.

3.2. Частина друга. Обчислення визначеного інтегралу

Вхідні дані ЛР 2. Частина 2

Задані користувачем початкові дані – це ліва та права межі інтегрування для знаходження значення визначеного інтеграла, а також кількість проміжків розбиття, на які розділяється інтервал інтегрування, також задається похибка обчислення інтеграла (всі значення вводяться з клавіатури). Варіанти завдань надані у табл. 6.

Вихідні дані ЛР 2. Частина 2

Початкові дані, значення інтеграла при заданій кількості проміжків, кількість проміжків, при якої похибка обчислення інтеграла не перевищує заданого значення.

Завдання

1. Вивчити методи чисельного обчислення визначеного інтегралу, графічне пояснення алгоритмів розрахунку інтеграла [1-5].

2. Скласти алгоритми обрахунку визначеного інтеграла кожним методом. Під час складення алгоритму можна використовувати підходи, що визначені у ЛР1. Тобто спочатку можна створити програму, потім БС алгоритму, або навпаки.

Таблиця 6 — Варіанти завдань

Варіант	Інтеграл	Варіант	Інтеграл
1	$\int_0^1 \frac{1}{4+x^2}dx$	2	$\int_0^1 \frac{xdx}{(x+2)^2}$
3	$\int_1^2 e^x dx$	4	$\int_{-3}^{-2} \frac{dx}{x^2-1}$
5	$\int_0^1 x\cdot\sin^2(x)dx$	6	$\int_{-3}^{0} \frac{dx}{\sqrt{25+3\cdot x}}$
7	$\int_0^1 x^2\sqrt{1+x^3}dx$	8	$\int_{0.5}^{4} \frac{1+\sqrt{x}}{x^2}dx$
9	$\int_1^2 x\cdot e^{x^2}dx$	10	$\int_{-4.5}^{10} x^2\sin(x)dx$
11	$\int_0^1 x^2\cdot e^{2\cdot x}dx$	12	$\int_{-1.4}^{\pi}(x^2+2\cdot x)dx$
13	$\int_{-1}^{1} \frac{dx}{(1.1+x)^2}$	14	$\int_0^3(\sqrt{2\cdot x^3}+\sqrt{x})dx$
15	$\int_1^3 \frac{dx}{2x^2+3x-2}$	16	$\int_0^1 \frac{dx}{1+x}$
17	$\int_0^5 \frac{dx}{2\cdot x+1.5}$	18	$\int_{-2}^{1} x^2 dx$
19	$\int_1^2 \frac{x}{2}\sqrt{x^2-1}dx$	20	$\int_{-1}^{3}(x^4+3\cdot x)dx$
21	$\int_{0.5}^{4} \frac{\ln(x)}{x}dx$	22	$\int_1^3(\sqrt{x}+x^2)dx$
23	$\int_1^3 \sqrt{x+1}\cdot dx$	24	$\int_0^1(2^x+x^2)dx$

Таблиця 6 — Варіанти завдань (продовження)

Варіант	Інтеграл	Варіант	Інтеграл
25	$\int_0^1 \frac{x^3 \cdot dx}{x^8+1}$	26	$\int_{-4}^{1} 2^{\lg(x)} dx$
27	$\int_3^{4.7} \frac{dx}{x^3-3x+2}$	28	$\int_{-\pi/3}^{-\pi/6} \frac{dx}{1+\mathrm{tg}(x)}$
29	$\int_0^1 \frac{dx}{x^2+5}$	30	$\int_{\frac{\pi}{6}}^{3\pi} \left(\sin(7x)+\sqrt{x}\right)dx$

3. Скласти програму та БС алгоритму обчислення інтеграла чотирма методами, а саме: метод прямокутників (лівих, правих), метод трапецій, метод парабол. У вихідному коді програми підінтегральний вираз розраховувати у окремій функції, що написана на мові програмування С. Методи розрахунків також реалізувати у вигляді окремих функцій на мові С.

4. Налагодити програму. У якості підінтегральної функції обрати функцію відповідно варіанту з табл. 6.

5. Виконати розрахунки для функції за варіантом при різних значеннях кількості проміжків *n* (наприклад *n*=10, *n*=100, *n*=1000, *n*=10000), а отримані значення знанести в таблицю результатів в протоколі лабораторної роботи.

6. Виконати розрахунок кількості проміжків *N*, при якому значення інтегралів *I1* (при кількості проміжків *N*) та значення інтегралу *I2* (при кількості проміжків *N+2*) відрізняються не більше ніж на задану похибку ε (похибку обирати в межах 0,00001 ≤ε≤0,001).

7. Побудувати таблицю з результатами обрахунків заданого інтеграла всіма методами при 4-5 кількостях проміжків. Побудувати графік залежності значення інтеграла від кількості проміжків розбиття для одного з методів (розрахувати 8-10 точок).

8. Зробити висновки щодо точності та швидкодії (кількості ітерацій) кожного з методів. Підготувати звіт. Вихідний код і звіт розташувати аналогічно як показано у ЛР № 1.

Програма проведення експерименту ЛР 2. Частина 2

Далі наведений варіант програми проведення експерименту.

1. Написати функцію мови програмування С, що обчислює значення інтегрального виразу. Функцію обрати відповідно до номеру варіанту. Протестувати функцію, переконатися у її правильній роботі.

2. Написати код, що реалізує текстовий інтерфейс, у якому обираються методи обчислення визначеного інтегралу, а саме: метод прямокутників, метод трапецій, метод парабол. Передбачити введення кількості проміжків.

3. Доповнити код п.2, реалізувати обчислення інтегралу чотирма різними методами, для цього написати чотири окремі функції, кожна з яких реалізує відповідний метод. Тобто, реалізувати кожний метод розрахунку інтегралу у вигляді функції мови програмування С. Налагодити і протестувати рішення.

4. Додати у програму новий функціонал у якому розраховується значення інтегралу, обраного відповідно до варіанту для двох значень кількості відрізків.

4.1. У першому розрахунку обчислюється значення інтегралу для кількості відрізків *N*, результат записується до змінної *I1*.

4.2. Далі визначити значення інтегралу при кількості відрізків *N+2*, результат записується до змінної *I2*.

4.3. Обчислити значення змінної, відповідно формули

$$Delta = | I1 – I2 |.$$

4.4. Написати додатковий код до програми, що обчислює значення *Delta* на протязі декількох кроків. На кожному кроці відбувається збільшення кількості відрізків *N* на *2*, поки значення *Delta* не буде у межах заданої похибки ε (похибку обирати в межах $0{,}00001 \le \varepsilon \le 0{,}001$).

5. Побудувати таблицю з результатами обрахунків заданого інтеграла всіма методами при 4-5 кількостях проміжків і графік залежності значення

інтеграла від кількості відрізків розбиття для одного з методів (розрахувати 8-10 точок). Вигляд таблиці показаний на рис. 11.

6. Побудувати блок-схему алгоритму отриманого програмного рішення.

7. Вихідний код і проект розташувати на https://github.com/.

8. Зробити висновки щодо точності та швидкодії (кількості ітерацій) кожного з методів. Оформити звіт до другої частини лабораторної роботи відповідно до наданих дали вимог. Звіт також розташувати на https://github.com/.

Метод	*n*=...	*n*=...	*n*=...	*n*=...	*N*
Лівих прямокутників					
Правих прямокутників					
Трапецій					
Парабол					

Рис. 11 — Приклад оформлення результатів підрахунку значень визначеного інтеграла для різної кількості проміжків розбиття інтервалу інтегрування

Теоретичні відомості ЛР 2. Частина 2

Задача чисельного інтегрування полягає в обчисленні визначеного інтеграла у випадках, коли аналітичне обчислення неможливе або дуже складне. Існують спеціалізовані програмні пакети для математиків, що дозволяють вирішити цю задачу. Крім математичних пакетів, рішення задачі можливо з використанням різних мов програмування, у тому числі і з використанням мови С. Методи чисельного обчислення інтеграла засновані на тому, що в якості наближеного значення інтеграла береться значення інтеграла для функції, яка інтерполює $f(x)$ по певних точках розбиття на відрізку $[a,b]$. На рис. 12 показані формули для чисельного обчислення визначеного інтегралу, а саме: на рис. 12.а показаний метод лівих прямокутників; на рис. 12.б показаний

метод правих прямокутників; на рис. 12.в – метод трапецій; на рис. 12.г – метод Сімпсона (метод парабол), працює лише для парної кількості проміжків.

Основна теорія щодо чисельного обчислення визначених інтегралів та її алгоритмізація показана у третьому розділі роботи [5].

Постановка задачі. Припустимо, маємо функцію $f(x)$, де x змінюється на проміжку $[a, b]$. Треба обчислити значення визначеного інтегралу.

Припустимо, що для неперервної $f(x)$ на проміжку $[a, b]$ можна віднайти первісну функцію $F(x)$:

$$F(x) = \int_a^b f(x)\ dx.$$

У такому випадку визначений інтеграл легко обчислити за формулою Ньютона–Лейбніца, що показана далі [5]:

$$\int_a^b f(x)\ dx = F(b) - F(a).$$

В іншому випадку застосовують методи чисельного інтегрування, що базуються на тому, що значення інтеграла дорівнює площі фігури, розміщеної поміж лінією графіка функції та віссю абсцис. Згідно з цими методами, проміжок інтегрування $[a, b]$ треба поділити на кілька відрізків (наприклад, поділити на n відрізків), та замінити площу S на суму площ елементарний фігур s_i (наприклад, суму площ утворених трапецій), що інтерполюють площу під кирвою функції $f(x)$ [5]:

$$S = \sum_{i=0}^{n-1} s_i.$$

Для цього варіанту рішення розроблені методи обчислення визначених інтегралів. Узагальнені формули для певних методів розрахунку представлені на рис. 12.

В лабораторній роботі пропонується дослідити відповідні методи та

перевірити результати обрахунку, використовуючи для цього спочатку якусь просту функцію, для якої можно знайти аналітичний вираз, а потім перевірити отримані результати із результатами роботи програми, яка реалізує запропоновані чисельні методи.

$$\int_a^b f(x)dx \approx h\cdot\left(f(x_0)+f(x_1)+\ldots+f(x_{n-1})\right)=h\cdot\sum_{k=0}^{n-1}f(x_k)$$

а)

$$\int_a^b f(x)dx \approx h\cdot\left(f(x_1)+f(x_2)+\ldots+f(x_n)\right)=h\cdot\sum_{k=1}^{n}f(x_k)$$

б)

$$\int_a^b f(x)dx \approx h\cdot\left(\frac{f(x_0)+f(x_1)}{2}+\frac{f(x_1)+f(x_2)}{2}+\ldots+\frac{f(x_{n-1})+f(x_n)}{2}\right)=$$

$$=h\cdot\sum_{k=0}^{n-1}\frac{f(x_k)+f(x_{k+1})}{2}=h\cdot\left(\frac{f(x_0)}{2}+\sum_{k=1}^{n-1}f(x_k)+\frac{f(x_n)}{2}\right)$$

в)

$$\int_a^b f(x)dx \approx \frac{h}{3}\cdot\left(f(x_0)+4\sum_{k=1,3,5\ldots}^{n-1}f(x_k)+2\sum_{k=2,4,6,\ldots}^{n-2}f(x_k)+f(x_n)\right)$$

г)

Рис. 12 — Формули для чисельного обчислення визначених інтегралів

Кінцеве значення чисельного розрахунку інтегралу містить певну похибку, позначимо її $R(x)$, тоді справедлива формула [5]:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1}s_i + R(x).$$

Залежно від того, якими методами здійснюється інтерполяція $f(x)$, існують різні методи (формули) обчислення визначених інтегралів. Найбільш відомі в інженерній практиці методи чисельного інтегрування: прямокутників, трапецій та Сімпсона, що розкриті у [5].

У якості прикладу розглянемо метод трапецій на основі теоретичного матеріалу [5]. У підґрунтя формули трапецій покладено заміну кривої підінтегральної функції на ламану лінію, як це показано на рис. 13. Відповідно до рис. 13, відрізок у межах інтегрування [*a*, *b*] розділений на *n* рівних частин. Позначимо довжину відрізків *h* і знайдемо її за формулою $h = (b - a) / n$.

Далі сполучимо прямими лініями значення функцій на кінцях відрізків, як це показано на рис. 13. Таким чином, значення визначеного інтегралу наближено відповідає сумі площин *n* трапецій, що утворені на відрізку [*a*, *b*].

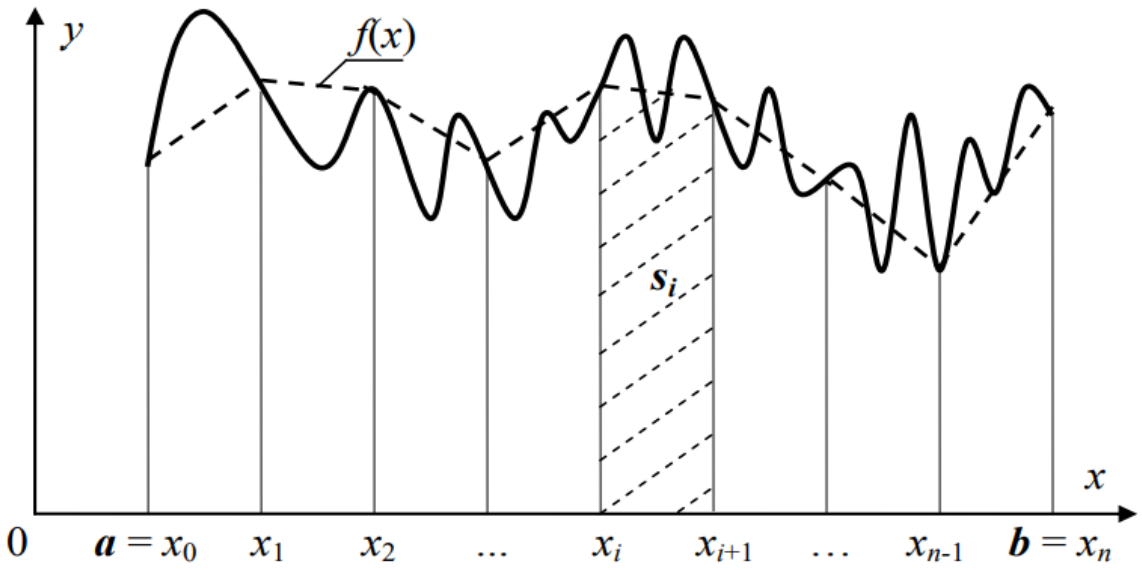


Рис. 13 — Основи метода трапецій [5]

Площу однієї такої трапеції (наприклад на рис. 13 вона відокремлена штриховими лініями) можна обчислити за формулою [5]:

$$s_i = \frac{1}{2} h\left(f(x_i) + f(x_{i+1})\right),$$

а загальна площа *S* всіх *n* трапецій і відповідно наближене значення інтегралу дорівнює [5]:

$$S = \sum_{i=0}^{n-1} s_i = \sum_{i=0}^{n-1} \frac{h}{2}\left(f(x_i) + f(x_{i+1})\right) = \frac{h}{2}\left(f(x_0) + f(x_n) + 2\sum_{i=1}^{n-1} f(x_i)\right).$$

Якщо підставити граничні значення проміжку обчислення інтеграла, то

формула набуде остаточного вигляду [5]:

$$S = \frac{h}{2}\left(f(a) + f(b) + 2\sum_{i=1}^{n-1} f(x_i) \right).$$

Значення похибки інтегрування можна оцінити за формулою [5]:

$$R(x) = \frac{h^2 (b-a)}{12} M_2,$$

де

$$M_2 = \max |f''(\xi)|, \ \xi \in [a, b] .$$

Для того щоб похибка не перевищувала задане значення ε, крок інтегрування слід обирати з умови [5]:

$$h \le \sqrt{\frac{12\varepsilon}{(b-a)M_2}}.$$

Узагальнений алгоритм для реалізації математичної теорії показаний далі на рис. 14 і може бути реалізований будь якою мовою програмування, у тому числі і мовою С [5].

Опишемо цей алгоритм у контексті використання мови програмування С. Оголошуємо змінні *a, b, h, n, S*, тип даних обираємо double. На першому етапі умовно розділяємо відрізок [*a, b*] на *n* рівних частин і визначаємо *h* = (*b* – *a*) / *n*.

Далі ініціалізуємо змінну для зберігання значення інтегралу — записуємо до змінної *S* нуль. Організуємо цикл для розрахунку інтегралу. Цикл буде виконуватися від одиничного значення лічильника *i* до *n-1*. Під час цього відбувається розрахунок *S*. Після закінчення циклу відбувається остаточний розрахунок значення інтегралу за формулою рис. 12 [5].

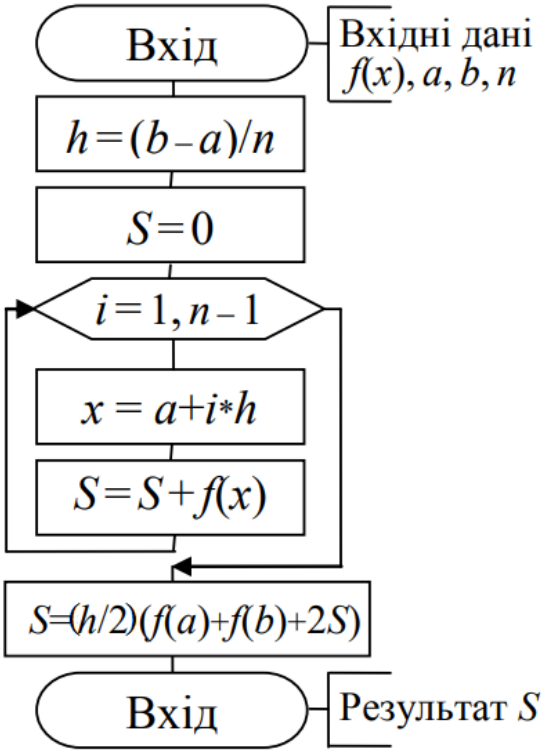


Рис. 14 — Узагальнена блок-схема алгоритму методу трапецій [5]

Далі надається вихідний код програми, що може бути корисним як прототип для реалізації лабораторної роботи. Він містить чотири функції, а саме: три функції для обчислення інтегралів різними методами і одну функцію, що містить інтегральний вираз. У вихідному коді окремо реалізовані прототипи функцій і окремо відбувається імплементація. Імплементація здійснена тільки для однієї функції, інші треба написати самостійно.

Програма працює у нескінченному циклі і дає можливість здійснити багаторазове тестування отриманого рішення. У двох варіантах (1,3) просто здійснюється обчислення інтегралу і вивід результатів на консоль. У другому варіанти здійснюється підбір рішення. Даний прототип вимагає доопрацювання.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
double num_comput_integral_l_re(double left_boundary_a,
                        double right_boundary_b,  unsigned int intervals);
double num_comput_integral_r_re(double left_boundary_a,
                        double right_boundary_b,  unsigned int intervals);
double num_comput_integral_Simps (double left_boundary_a,
                         double right_boundary_b, unsigned int intervals);
double integrand_expression( double x );

int main()
{
   double left_boundary_a=0, right_boundary_b=0;
   double measurement_error=0, I1=0, I2=0;
   int intervals, var, i;
   double integral_s=0;

while(1)
   {
   printf("\n\tEnter the left boundary \n  X(first)=");
   scanf("%lf", &left_boundary_a);
   printf("\n\tEnter the right boundary \n  X(last)=");
   scanf("%lf", &right_boundary_b);
   do{
                printf("\tEnter the number of partition intervals (N>0)\nN=");
                scanf("%u", &intervals);
    }while(intervals <= 0);

 printf("\n\tEnter the measurment error of integration\n  Measurment error=");
 scanf("%lf", &measurement_error);
 do
  {
                printf("\nChoose the method of calculating:\n");
                printf("\t1. By Left Rectangles :\n");
                printf("\t2. By Right Rectangles:\n");
```

```
                printf("\t4. By ntegral_Simps's method (parabola method):\n");
                scanf("%u", &var);
                    if (var!=1 && var!=2 &&var!=3 )
                            printf("\nYou are mistaken\n");
    }while (var!=1 && var!=2 && var!=3 );

system("cls");
  switch(var)
      {
      case 1:
        {
          integral_s = num_comput_integral_l_re(left_boundary_a,
                                    right_boundary_b, intervals);
          printf("\n\n\t*Left Rectangles method*\n");
          printf("\n\ta = %.2lf    \n\tb = %.2lf   \n\tIntegral = %.8lf   \n\tN = %d",
                left_boundary_a, right_boundary_b,  integral_s, intervals);
        }
      break;

      case 2:
        {
          printf("\n\n\t======*Right Rectangles method*======\n");
          I1 = num_comput_integral_r_re(left_boundary_a,
                            right_boundary_b,  intervals);
          for (i = 0; i < 2; i++) {
             if (i == 1){
                intervals += 2;
                I2 = num_comput_integral_r_re(left_boundary_a,
                                    right_boundary_b,  intervals);
                if (fabs(I1 - I2) <= measurement_error) {
                   break;
                }
             }
             else {
```

```
                    i = 0;
                }
            }
            printf("\n\ta = %.2lf    \n\tb = %.2lf   \n\tIntegral = %.8lf   \n\tN = %d",
                left_boundary_a, right_boundary_b, I2, intervals);
        }
        break;
        case 3:
        {
            integral_s = num_comput_integral_Simps(left_boundary_a,
                                    right_boundary_b,  intervals);
            printf("\n\n\t======*num_comput_integral_Simps's method *======\n");
            printf("\n\ta = %.2lf    \n\tb = %.2lf   \n\tIntegral = %.8lf   \n\tN = %d",
                left_boundary_a, right_boundary_b,  integral_s, intervals);
        }
        }
}
    return 0;
}

// Опис функцій, які повинні реалізувати алгоритм обчислення визначеного
 // інтегралу відповідним методом.
// В якості прикладу, наводиться лише реалізація одного методу.
// Для інших методів представлено лише заголовки функцій.
// Опис функцій повинен бути виконаний студентом самостійно.


double num_comput_integral_I_re(double left_boundary_a,
                        double right_boundary_b,  unsigned int intervals)
{
    double integral_s=0, x=0, h;
    unsigned int i;
    h = ( right_boundary_b - left_boundary_a ) / intervals;
    x = left_boundary_a;
```

```
    for (i = 0;  i < intervals;  i++ ){
      integral_s += integrand_expression(x);
      x += h;
    }
    return integral_s*h;
}

double num_comput_integral_r_re(double left_boundary_a,
                        double right_boundary_b,  unsigned int n)
{
   ;
}

double num_comput_integral_Simps (double left_boundary_a,
                          double right_boundary_b,  unsigned int n)
{
  ;
}

double integrand_expression( double x )  // основна функція
{
    return  pow(x,2) + 2*x;
}
```

Наведемо узагальнений приклад блок-схеми алгоритму для рішення завдання другої частини ЛР № 2.

Блок-схема зображена на рис. 15. У алгоритмі у блоці 2 відбувається оголошення і ініціалізація змінних. Далі відбувається виведення на консоль або термінал запрошення користувача програми на введення варіанту виконання (блок 3). Далі у блоці 4 відбувається введення варіанту користувачем програми. Наступним кроком визначається гілка, що реалізує знаходження значення інтегралу відповідним методом.

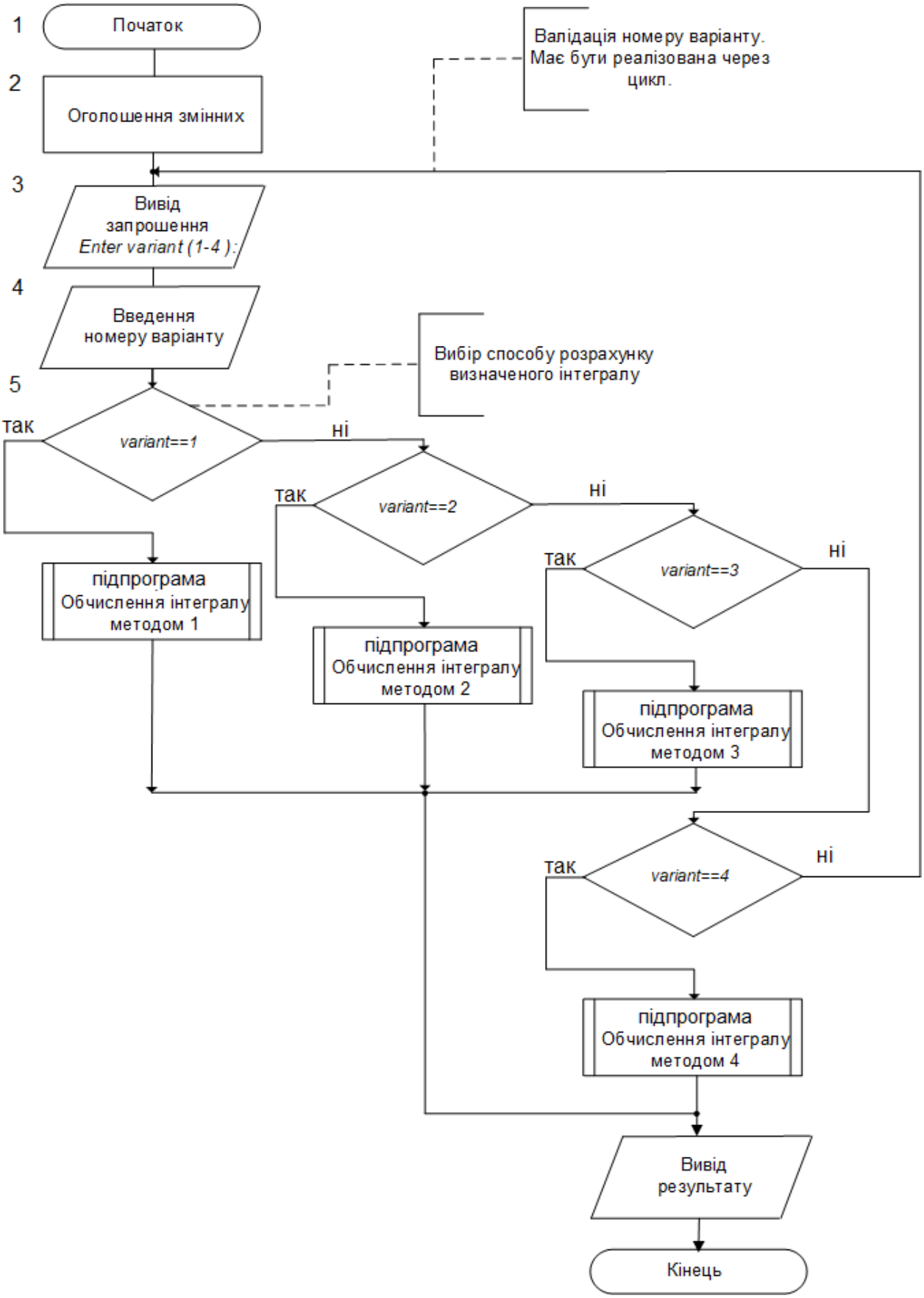


Рис. 15 — Узагальнена блок-схема алгоритму для другої частини ЛР 2

Кожний метод розрахунку реалізований у вигляді відповідної підпрограми на блок-схемі алгоритму і цій підпрограмі відповідає певна функція. Використовуючи вказану блок-схему і прототип вихідного коду можна побудувати рішення, що сформульовано у завданні. Основними елементами програми є текстовий інтерфейс програми і функції розрахунку чисельного значення інтегралу відповідним способом. Кількість ітерацій, що потрібна для розрахунку обчислюється у ході виконання програми.

3.3. Контрольні запитання до лабораторної роботи № 2

1. Що необхідно зробити, щоб мати змогу використовувати відповідні функції стандарної бібліотеки?
2. Навіщо потрібні прототипи функцій?
3. Чи ігноруються компілятором імена параметрів в прототипах функцій?
4. Чим відрізняються формальні та фактичні параметри?
5. Наведіть приклад опису функції, що знаходить добуток трьох параметрів, які передаються у функцію, і результат обрахунку повертається в точку виклику даної функції і присвоюється відповідній змінній.
6. Чи можна створити функцію, яка не отримує жодних параметрів і не повертає ніякого значення (функція має тип void)? Якщо таку функцію створити можна, то наведіть приклад опису такої функції.
7. Яке призначення оператора return в функції типу void ?
8. Який заголовочний файл необхідно вставити в текст програми, щоб мати змогу використовувати функції *printf*() та *scanf*() ?
9. Чи потрібно вставляти в текст програми заголовочний файл *math.h,* щоб мати змогу використовувати операції множення, ділення, додавання, віднімання в арифметичних виразах?
10. Які можливі результати можна отримати при обрахунку логічного виразу? Поясніть як працюють операції *логічне І* та *логічне АБО.*

4. ЛАБОРАТОРНА РОБОТА № 3 «РОЗВ'ЯЗАННЯ НЕЛІНІЙНИХ РІВНЯНЬ»

Мета роботи: розробка програми розв’язання нелінійних рівнянь. Також додатковою метою лабораторної роботи є закріплення знань, умінь та навичок з технології розроблення програмного забезпечення з використанням мови програмування С.

Лабораторна робота складається з двох частин. Перша частина передбачає виконання завдання для підготовки другої частини лабораторної роботи і орієнтована на виконання первинного дослідження функції нелінійного рівняння. Рівень складності лабораторної роботи першої частини відповідає рівню складності лабораторної роботи № 4-5 [2]. Після виконання і захисту першої частини лабораторної роботи вважається захищеною у цілому. Максимальна кількість балів, яка може бути нарахована за першу частину лабораторної роботи становить половину від максимальної кількості балів, що нараховується за виконання основного завдання (другої частини) лабораторної роботи. Друга частина передбачає виконання повного завдання відповідно до силабусу даної дисципліни. За другу частину при успішному виконанні та захисті нараховується максимальна оцінка з урахуванням штрафних і додаткових балів.

Можливе виконанння підготовленими студентами лише другої частини завдання лабораторної роботи без зменшення максимальної кількості балів.

4.1. Частина перша. Дослідження функції

Вхідні дані ЛР 3. Частина 1

Функція для дослідження $f(x)$ обирається відповідно до варіанту з табл. 1, другої частини лабораторної роботи № 1. Початкове та кінцеве значення

аргументів для $f(x)$ вводяться з клавіатури. Також з клавіатури вводиться кількість значень аргументів (кількість точок), для яких знаходиться значення функції $f(x)$. Значення функції при аргументах із сформованого проміжку будуть виводитися на консоль.

Вихідні дані ЛР 3. Частина 1

Проект IDE Code::Blocks та вихідний код програми мовою С. Вихідний код має містити функції мови програмування С для розрахунку вихідної математичної функції та її похідної.

Таблиця аргументів функції $f(x)$ та її значень (складається зі стовпчиків, що умовно можна назвати "номер точки", " значення аргументу" та "значення функції"). Вихідний код мови програмування С для розрахунку таблиці.

Таблиця аргументів функції $f(x)$ та значень її похідної $f(x)/dx$ (складається зі стовпчиків, що умовно можна назвати "номер точки", "значення аргумент" та "значення похідної функції"). Вихідний код мови програмування С для розрахунку таблиці.

Завдання

Перша частина лабораторної роботи призначена для підготовки даних для виконання другого етапу лабораторної роботи № 3. За результатами треба провести дослідження $f(x)$ і визначити проміжки ізоляції $[\alpha, \beta]$, на якому є один і лише один розв'язок ξ рівняння.

1. У якості індивідуального завдання, в першій частині ЛР 1 необхідно написати код, що реалізує текстовий інтерфейс, у якому обирається два варіанта введення початкових даних.

1.1 Перший варіант початкових даних передбачає введення:

- початкового значення аргументу (*X1*),
- кінцевого значення аргументу (*X2*),
- кількість точок в таблиці (*N*).

1.2 Другий варіант початкових даних передбачає введення:

- початкового значення аргументу (*X1*),
- кінцевого значення аргументу (*X2*),
- крок зміни аргументу (*delta*).

2. Для збереження *X1, X2, delta* оголосити змінні, що мають тип *double*, для збереження *N* оголосити змінну, що має тип *unsigned int* (або *int*).

3. У програмі передбачити валідацію даних (контроль допустимого значення під час введення даних).

4. Перед виводом таблиці на екран необхідно вивести введені початкові дані. Таблицю накреслити за допомогою символів псевдографіки або інших символів. Ширина кожного стовпця береться довільною.

5. Приклад вигляду екрану показаний на рис. 4 (лабораторна робота № 1).

6. Реалізувати програмно вивід на консоль (термінал) таблиці функції та її похідної.

7. Звіт до першої частини лабораторної роботи оформити відповідно до вказаних вимог. Звіт також розташувати на https://github.com/.

Теоретичні відомості ЛР 3. Частина 1

При реалізації першої частини ЛР 3 необхідно використовувати циклічні оператори (*while, do…while, for*), а також необхідно розробити та реалізувати власні функції, які вирішують відповідні частини завдання.

Цикл *do…while* — цикл з постумовою. Спочатку виконується тіло циклу (виконується ітерація), і лише після цього виконується перевірка умови, і в залежності від результату логічного виразу, який записується в умові, відбувається продовження циклу, або ж відбувається вихід із циклу.

Цикл *while* — цикл з передумовою. Спочатку виконується перевірка умови, і в залежності від результату логічного виразу, який записаний в умові, відбувається виконання тіла циклу, або ж вихід із циклу.

Крім циклічних операторів, в лабораторній роботі необхідно також написати власні функції мови програмування С, що реалізують певні алгоритми. При використанні власної функції мови програмування С необхідно записати прототип функції, а також виконати опис функції.

Функції, які мають бути реалізовані в лабораторній роботі, поділяються на функції мови програмування, що *повертають* значення, та функції, які *не повертають* значення (це функції типу *void*).

У якості прикладу для пояснення роботи із функціями, розглянемо функцію на мові програмування С, яка підраховує середній бал успішності студента. Ця функція буде називатися *averageGrade*. При цьому функція отримує аргумент — це кількість предметів для яких буде порахований середній бал. Отримавши цей аргумент, передбачається введення з клавіатури балів по кожному предмету. На етапі введення балів, необхідно передбачити можливість перевірки потрапляння введеного балу в проміжок 60…100, і якщо введений бал виходить за ці рамки, необхідно забезпечити можливість користувачу ввести правильний бал повторно. Після введених балів, функція обраховує середній бал успішності і повертає це значення в точку виклику, де повернений середній бал присвоюється відповідний змінній.

Інша функція, яка має бути реалізована в програмі — це функція типу *void*, яка отримує в якості аргументу середній бал успішності (ця функція буде називатися *printGrade*). Функція повинна виводити середній бал на екран монітора.

Враховуючи, що в програмі необхідно буде використати бібліотечні функції по введенню та виведенню значень, а також будуть використані бібліотечні функції *system* та *exit*, тому необхідно за допомогою директиви препроцесора *include* вставити в програму відповідні заголовочні файли:

```
#include <stdio.h>

#include <stdlib.h>
```

Також необхідно написати прототипи тих двох власних функцій на мові програмування С, які будуть описуватися в програмі і виконувати вищезазначені дії. Для цих функцій обрано імена *averageGrade* та *printGrade*. Вказані функції будуть отримувати відповідні параметри. Прототипи для цих двох функцій можуть бути записані так:

```
double averageGrade(unsigned int number_of_subjects);

void printGrade( double ave_grade );
```

Функція *averageGrade* отримує один параметр типу *unsigned int*. Цей параметр буде відповідати за кількість предметів, для кожного з яких буде передбачатися введення балу. Функція буде розраховувати середній бал і повертати це значення в місце виклику функції в програмі.

Функція *printGrade* має тип даних *void*. Ця функція не повертає ніяких значень. Функція отримує параметр типу *double* і цей параметр містить середній бал, який був раніше обрахований в програмі. Функція *printGrade* повинна вивести на екран значення середнього балу, а також супроводжувати виведення значення певними коментарями. Виведення середнього балу відбувається із одним знаком після коми. Виклик функцій *averageGrade* та *printGrade* виконується у функції *main*.

Також всередині функції *main* оголошуються змінні *subjects* та *mean*. Це локальні змінні. Якщо змінні оголошуються перед *main*, то такі змінні називаються глобальними змінними. Глобальні змінні доступні у будь-якому місці програми.

Глобальні змінні для виконання практичного завдання лабораторних робіт 1–7 використовувати не треба!

Справа у тому, що для налагодження такої програми, для контролю глобальної змінної доведеться переглянути повністю вихідний код, щоб простежити логіку виконання і зміни значень змінних у програмі. Також глобальні змінні роблять програму менш модульною та гнучкою. Функція, яка

використовує тільки свої параметри і не має побічних ефектів, є ідеальною в плані модульності.

Модульність допомагає зрозуміти структуру програми, що вона робить і як можна повторно використовувати певні частини коду в іншій програмі. Глобальні змінні значно зменшують цю можливість.

Змінна *subjects* є локальною і зберігає введене з клавіатури значення кількості предметів. Для заданої кількості предметів необхідно буде ввести бали і розрахувати середнє значення. Змінна *subjects* буде мати тип даних *unsigned int*.

Змінна *mean* буде зберігати значення середнього балу. Змінна mean буде отримувати значення, яке буде повертатися функцією *averageGrade*. Змінна *mean* буде мати тип даних *double*. Необхідно передбачити, що у випадку коли користувач на запит ввести кількість предметів для яких буде підраховуватися середній бал (це значення зберігається в змінній *subjects*) введе значення нуль, то програма має завершитися. Враховуючи, що при виведенні текстової інформації в командне вікно (в консоль) можуть некоректно виводитися літери українського алфавіту, тому текстові коментарі можна записувати на трансліті (написання українських слів літерами англійського алфавіту) або ж виводити текстові повідомлення англійською мовою. Таким чином, функція *main* може мати такий вигляд:

```
int main()
{
   unsigned int subjects;
   double mean;
   printf("Enter number of subjects. This value must be greater than ZERO.");
   printf("\nOr enter ZERO to exit this program");

   printf("\n\nsubjects= ");
```

```
    scanf("%u", &subjects);

    if(subjects == 0 )
        exit(0);

    mean = averageGrade( subjects );      // виклик ф-ції averageGrade. Ф-ція повертає
                                          // обраховане середнє значення

    printGrade( mean );                   // виклик функції printGrade. Функція має тип
                                          // даних void. Функція виводить на екран
                                          // значення розрахованого  середнього балу
    return 0;
}
```

Тепер необхідно виконати опис двох власних функцій. Перша функція *averageGrade* буде мати наступний заголовок:

```
double averageGrade(unsigned int  number_of_subjects)
```

Коли функція *averageGrade* буде викликана, відбудеться створення локальної змінної на ім’я *number_of_subjects* типу *unsigned int*, і ця змінна отримає значення параметру, який стоїть у виклику функції *averageGrade* всередині функції *main*. Таким чином, змінна *number_of_subjects* отримує ніби як копію значення змінної *subjects*, яка виступає аргументом функції *averageGrade* при її виклику у функції *main*.

Передбачається, що бали, які будуть вводитися мають бути в діапазоні 60…100. Якщо користувач буде намагатися вводити значення, що виходять за цей діапазон, програма повинна повторити запит на введення балу. Це може бути реалізовано за допомогою оператора циклу do…while. Один із варіантів

написання програмного коду, що реалізує такий функціонал, може бути представлений наступним чином:

```
do{
  printf("\ngrade #%u: ", i);
  scanf("%u", &grade);
}while( grade < 60  ||  grade > 100 );
```

де змінна *i* зберігає номер предмету для якого вводиться бал. Змінна *i* має тип даних *unsigned int*. А змінна *grade* зберігає значення балу. Ця змінна також має тип даних *unsigned int*. Наведена частина програмного коду буде призводити до виведення на екран повідомлення наступного виду (в наведеному прикладі вважається, що змінна *i*=2) :

```
grade #2:
```

та буде очікуватися введення балу з клавіатури. Якщо користувач введе значення, що виходить за діапазон 60…100, тоді відбудеться повторний запит, наприклад:

```
grade #2: 52
grade #2: 102
grade #2:
```

Після введення правильного значення, відбудеться перехід до введення балу по наступному предмету, наприклад:

```
grade #2: 52
grade #2: 102
grade #2: 88
grade #3:
```

Кількість предметів зберігається в змінній *number_of_subjects*. Таким чином, введення балів по кожному предмету необхідно реалізувати за допомогою циклічного оператора. Наприклад, за допомогою циклу *for*. Для підрахунку середнього балу, необхідно знайти суму всіх введених балів. Для збереження цієї суми використовується змінна *sum* типу *unsigned int*. Ця змінна оголошується всередині функції *averageGrade*. Частина програмного коду, яка буде відповідати за правильність введення балів і підрахунок суми балів, буде мати наступний вигляд:

```
for( i=1;   i <= number_of_subjects;   i++ ){
    do{
        printf("\ngrade #%u: ", i);
        scanf("%u", &grade);
    }while( grade < 60  ||  grade > 100 );
    sum = sum + grade;
}
```

Після підрахунку суми балів необхідно знайти середній бал. Середній бал буде обраховуватися як сума балів, яка поділена на кількість предметів. Обраховане середнє значення буде зберігатися в змінній *mean_value*. Ця змінна буде мати тип даних *double*. Це буде локальна змінна, що оголошується всередині функції *averageGrade*. Враховуючи, що сума балів та кількість предметів зберігаються в змінних типу *unsigned int*, то щоб при діленні була збережена дробова частина, необхідно виконати явне перетворення типів даних у виразі, в якому знаходиться середній бал:

```
mean_value = (double)sum / (double)number_of_subjects;
```

Значення змінної *mean_value* повертається за допомогою оператора *return* в місце виклику функції *averageGrade* в функції *main*. Повернене значення у

функції *main* присвоюється змінній *mean*. Опис функції *averageGrade*, яка підраховує середнє значення, а також містить текстові коментарі, які повинні дати зрозуміти користувачу, що від нього очікується, може мати наступний вигляд:

```
double averageGrade(unsigned int number_of_subjects)
{
   unsigned int i, grade, sum;
   double mean_value;
   system("cls");
   printf("Enter grades for each subject. Grade must be between 60...100.");
   printf("\nTotal number of grades equals to %u\n", number_of_subjects);
   sum = 0;
   for( i=1;   i <= number_of_subjects;   i++ ){
      do{
         printf("\ngrade #%u: ", i);
         scanf("%u", &grade);
      }while( grade < 60  ||  grade > 100 );
      sum = sum + grade;
   }
   mean_value = (double)sum / (double)number_of_subjects;
   return mean_value;
}
```

Інша функція — *printGrade*. Ця функція виводить значення середнього балу на екран. Вона має наступний заголовок:

```
void printGrade( double  ave_grade )
```

При виклику функції *printGrade* буде створена локальна змінна *ave_grade*, і цій змінній буде присвоєно значення аргументу, який записується у виклику функції *printGrade* всередині функції *main*. У виклику функції *printGrade* всередині функції *main* в якості аргументу виступає змінна *mean*. Таким чином, змінній *ave_grade* буде присвоєно значення змінної *mean*. Функція *printGrade* має тип даних *void*, тобто ця функція не повертає ніякого значення в місце виклику цієї функції.

Функція *printGrade* може мати наступний вигляд:

```
void printGrade( double ave_grade )
{
   system("cls");
   printf("\n");
   printf("Average grade: %.1lf\n", ave_grade);
}
```

Функція очищує екран за допомогою команди: *system*("*cls*");

Потім переводить курсор на новий рядок за допомогою команди: *printf*("\n");

Після цього виводить на екран значення середнього балу. При цьому значення виводиться із одним знаком після коми. Виведення на екран середнього балу виконується за допомогою команди: *printf*("*Average grade: %.1lf\n*", *ave_grade*). Загалом, весь текст програми може мати наступний вигляд:

```
#include <stdio.h>
#include <stdlib.h>

double averageGrade(unsigned int number_of_subjects);
void printGrade( double ave_grade );

int main()
{
   unsigned int subjects;
   double mean;
```

```
    printf("Enter number of subjects. This value must be greater than ZERO.");
    printf("\nOr enter ZERO to exit this program");
    printf("\n\nsubjects= ");
    scanf("%u", &subjects);

    if(subjects == 0 )
        exit(0);

    mean = averageGrade( subjects );

    printGrade( mean );

    return 0;
}

//------------ Опис функції, яка знаходить середній бал ------------

double averageGrade(unsigned int number_of_subjects)

{

    unsigned int i, grade, sum;
    double mean_value;

    system("cls");
    printf("Enter grades for each subject. Grade must be between 60...100.");
    printf("\nTotal number of grades equals to %u\n", number_of_subjects);

    sum = 0;

    for( i=1;   i <= number_of_subjects;   i++ ){

        do{
            printf("\ngrade #%u: ", i);
            scanf("%u", &grade);
        }while( grade < 60  ||  grade > 100 );

        sum = sum + grade;

    }

    mean_value = (double)sum / (double)number_of_subjects;

    return mean_value;
}
```

```
//---------- Опис функції, яка виводить середній бал на екран ----------

void printGrade( double ave_grade )
{
   system("cls");
   printf("\n");
   printf("Average grade: %.1lf\n", ave_grade);
}
```

Розібравшись із особливостями роботи з функціями за допомогою представленого вище прикладу, необхідно переходити до реалізації завдання лабораторної роботи.

У якості підготовки до виконання другого частини лабораторної роботи № 3 доцільно провести дослідження похідної функції, спираючись на результати ЛР 1. Для цього потрібно доповнити код другої частини ЛР 1 і розрахувати значення похідної у діапазоні значення аргументу (*X1…X2*) на таких самих інтервалах як і у ЛР 1.

Для розробки програми знаходження кореня нелінійного рівняння потрібно визначитися із теорією чисельного рішення нелінійних рівнянь. Розглянемо нелінійне рівняння в узагальненому вигляді і позначимо його так:

$$f(x) = 0 \qquad (3.1)$$

Якщо $f(x)$ – нелінійний алгебраїчний многочлен, то рівняння (3.1) називають алгебраїчним. Якщо $f(x)$ містить якісь спеціальні математичні функції (наприклад $\cos x$, $\sin x$, $\lg x$, e^x тощо), то рівняння називають трансцендентним. У багатьох випадках аналітичний розв'язок рівняння (3.1) є складною задачею. У теперішній час, коли доступними є обчислювальні потужності доцільно розв'язувати нелінійне рівняння (3.1) наближеними методами.

Наближений метод розв’язування нелінійного рівняння можна розділити на два етапи, а саме:

- визначення проміжків ізоляції з метою обчислення наближених значень розв’язків рівняння (у цьому разі говорять про початкове (чи нульове) наближення);
- уточнення наближеного значення розв’язку до заданого степеню точності (за заданого значення похибки). Значення змінних ξi (де i = 1, 2, ...), за яких виконується рівняння $f(\xi i) = 0$, називаються нулями функції $f(x)$, чи розв’язками рівняння (3.1). Рівняння (3.1) може мати один чи кілька розв’язків або не мати жодного.

На графіку функції розв’язок зображується як точка перетинання функції $f(x)$ та осі x (абсцис). На рис. 16.а зображено функцію, яка має три розв’язки, на рис. 16.б – функцію з одним розв’язком на проміжку [α, β]. Проміжок [α, β], на якому є один і лише один розв’язок ξ рівняння (3.1), називають проміжком ізоляції, а сам розв’язок називають ізольованим на заданому проміжку.

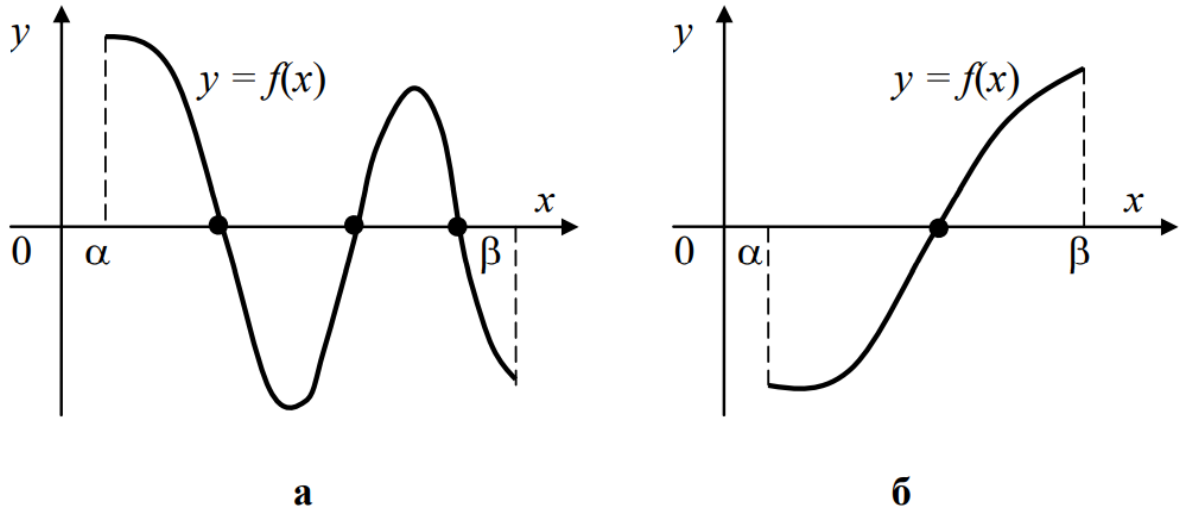


Рис. 16 — Геометрична інтерпретація розв’язків рівнянь [5]

Відомо, якщо функція $f(x)$ є неперервною на проміжку, а на кінцях проміжку набуває значення протилежних знаків, то всередині цього проміжку існує хоча б один розв’язок рівняння. Позначимо ці значення у наближеному вигляді як ξi (де i = 1, 2, ...), за яких виконується рівняння $f(\xi i) = 0$. Ці значення називаються нулями функції $f(x)$ і розв’язками рівняння (3.1).

Рівняння (3.1) може мати один або кілька розв'язків. Може бути такий випадок, що (3.1) не має жодного рішення. У більшості випадків для використання чисельних методів на першому кроці потрібно знайти саме відрізок [α, β]. Для визначення всіх проміжків ізоляції достатньо обчислити таблицю значень $f(x)$. Саме це завдання виконується у ЛР 1. Крім того для того, щоб довести, що на проміжку [α, β] існує тільки одне рішення рівняння потрібно переконатися, що похідна $f(x)/dx$ не змінює на відрізку [α, β] свій знак.

Теоретичні питання, що пов'язані з побудовою таблиці функції $f(x)$ розкриті у лабораторній роботі 1. Для побудови графіку похідної $f(x)/dx$ користуємося чисельними методами розрахунку похідної.

Виконання лабораторної роботи передбачає опанування використання циклічних операторів, а також написання власних функцій мови програмування С, які реалізують завдання. В лабораторній роботі використовуються циклічні оператори: *for*, *while*, *do…while*.

Далі наданий варіант програми, що дає можливість зробити порівняння чисельного і аналітичних методів розрахунку похідних.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <time.h>

double math_function(double x); //  2*pow(x,3)-0.2*pow(x,2)+0.5*x+3
double derivative_1_math_funct (double x);     // перша похідна функції
                                               // аналітична
double derivative_2_math_funct (double x);    //друга похідна функції
                                              //аналітична
double digit_deriv_1_math_left (double x, double h ); //перша похідна функції
                                                       // чисельна
```

```
double digit_deriv_1_math_r (double x, double h); // перша похідна функції
                                                  // чисельна

double digit_deriv_2_math_funct (double x, double h);    //друга похідна
                                                         // функції
void input_for_der(double *x, double *h);

void result(double f);


int main()
{
    for(;;)
    {
    double x, h;
    unsigned int var;
    printf("1 -  original der.\n2 -  der. digit left \n3 -  der. r c.\n");
    scanf("%d",&var);
        switch(var)
        {
            case 1:
                printf("Input x\n");
                scanf("%lf",&x);
                printf("x = %lf\n",x);
                result(derivative_1_math_funct(x));
            break;
            case 2:
                input_for_der(&x, &h);
                result(digit_deriv_1_math_left(x, h));
            break;
```

```
            case 3:
                input_for_der(&x, &h);
                result(digit_deriv_1_math_r(x, h));
            break;
        }
    }
    return 0;
}


double math_function(double x)//  2*pow(x,3)-0.2*pow(x,2)+0.5*x+3
{
    return 2*pow(x,3)-0.2*pow(x,2)+0.5*x+3;
}


double derivative_1_math_funct (double x)    //перша похідна функції
{
    return  6*pow(x,2)-0.4*x+0.5 ;
}


double derivative_2_math_funct (double x)    //друга похідна функції
{
    return 12*x-0.4;
}
```

```
double digit_deriv_1_math_left (double x, double h )
{   // Точка для похідної
    // Крок з яким обчислюється похідна
    // 1e-10 Derivative error
    if (h<1e-10)
    return  ((math_function(x) - math_function(x-h))/h);
    else
      printf("Too much value for h\n");
    return 1;
}

double digit_deriv_1_math_r (double x, double h)
{
    // 1e-10 Derivative error
    if (h<1e-10)
    return  ((math_function(x+h) - math_function(x))/h);
    else
      printf("Too much value for h\n");
    return 1;
}

double digit_deriv_2_math_funct (double x, double h) //друга похідна функції
{
;
  }
```

```
void input_for_der(double *x, double *h)
{
    printf("Input x, h 0.00000000001\n");
    scanf("%lf",x);
    scanf("%lf",h);
    printf("x = %lf\n",*x);
    printf("h  = %lf\n",*h);
}

void result(double f)
{
   printf("Result der 1 is = %lf\n",f);
}
```

Результат роботи програми показаний на рис. 17. Спочатку вводиться текстовий інтерфейс користувача. Після, за вибором користувача програми, відбувається обчислення похідної, що знайдена для функції аналітично у точці $x=2$. Далі на рис. 17 показано обчислення похідної чисельним методом також для точки $x=2$. Результати для двох варіантів показують практичне співпадіння значень похідної, що обчислена в точці аналітично і чисельним методом.

4.2. Частина друга. Розв'язання нелінійних рівнянь

Вхідні дані ЛР 3. Частина 2

Вираз для нелінійного рівняння обрати у таблиці 1, лабораторної роботи № 1, частина 2. Проміжки ізоляції [α, β], на якому є один і лише один розв’язок ξ рівняння, що визначний в першій частині цієї лабораторної роботи. Похибка обчислення і максимальна кількість ітерацій.

Вихідні дані ЛР 3. Частина 2

Проект IDE Code::Blocks, вихідний код програми мовою С. Результати перевірки проміжків ізоляції [α, β], що підтверджують наявність тільки одного кореня на ньому. Значення кореня, кількість ітерацій.

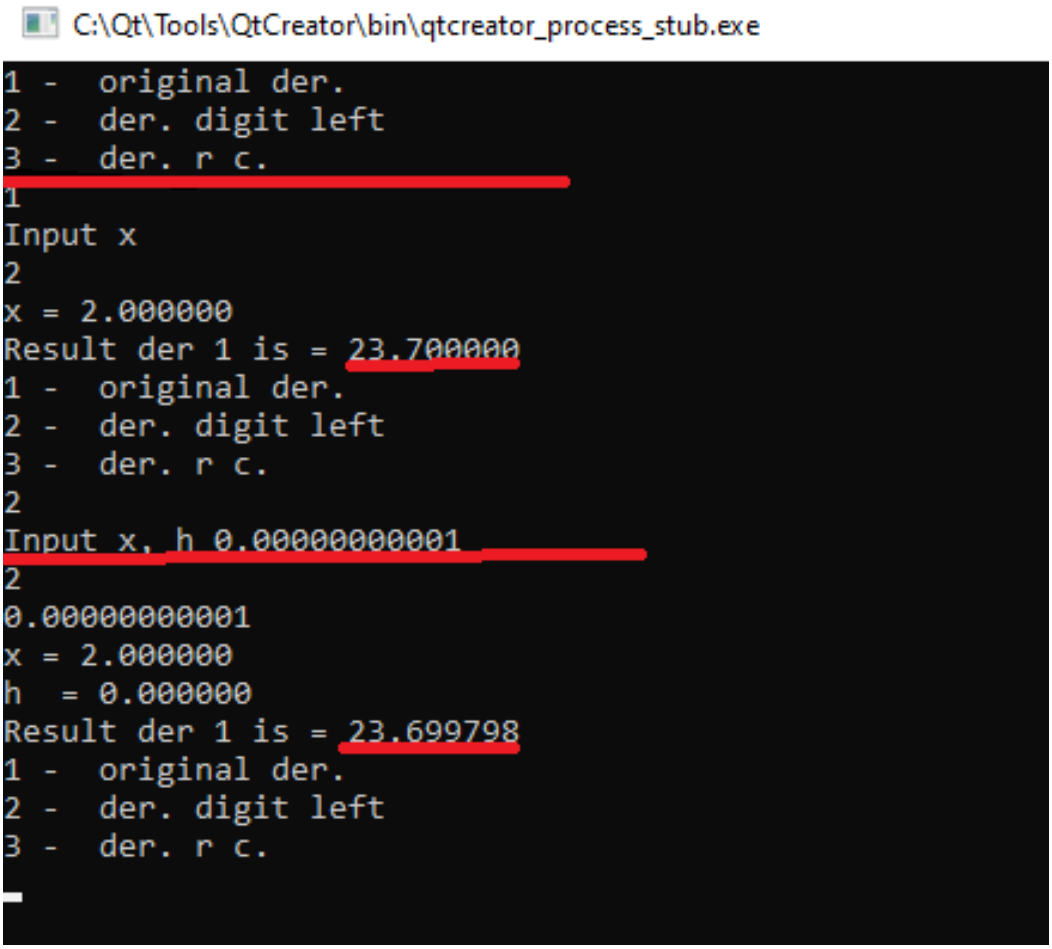


Рис. 17 — Результат роботи програму розрахунку похідної

Завдання

1. Вивчити методи розв’язання нелінійних рівнянь.

2. Скласти алгоритми програм розв’язання нелінійних рівнянь двома методами, методом хорд і методом половинного ділення. Реалізувати методи розв’язку для рівняння відповідно обраного варіанту. Реалізувати текстове меню користувача для вибору методу.

3. Написати програму розв’язання нелінійних рівнянь двома методами. В програмі передбачити:

- режим налагодження при якому відбувається виведення результатів на кожній ітерації (за вибором користувача);
- зупинку обчислення при перевищенні заданої кількості ітерацій з видачею інформації для прийняття рішення, що робити далі:
  а) продовжити з такою ж кількістю ітерацій;
  б) виконати програма до кінця, поки не будуть знайдено корені рівняння;
  в) вийти із програми, перед виходом вивівши на екран отриманий проміжний результат підрахунку;
- визначення затрат часу на пошук кореня, наприклад, за допомогою функцій *clock*() чи *time*();
- вивід на екран значення кореня та значення функції в цій точці.

4. Для налагодження програми взяти нелінійне рівняння з відомим розв’язком та провести тестування програми.

5. Дослідити збіжність методів від вибору початкових умов для нелінійного рівняння, обраного відповідно до варіанту.

6. Розрахувати значення похідної і підтвердити наявність одного рішення рівняння на проміжку. Протестувати програму для різних варіантів.

7. Підготувати звіт. Оприлюднити результати як показано у другій частині лабораторної роботи 1.

Програма проведення експерименту ЛР 3. Частина 2

1. Скласти алгоритми програм для розв’язання нелінійних рівнянь двома методами.

2. Написати текстовий інтерфейс програми, що дає можливість реалізувати програму двома методами.

3. Доповнити програму п.2 функціями мови програмування С, що дають можливість розв'язку нелінійного рівняння двома методами відповідно до завдання.

4. Доповнити програму п.3 режимом налагодження для виведення результатів на кожній ітерації.

5. Провести тестування отриманих рішень під час виконання програми.

6. Доповнити програму п.3 наступним функціоналом:

- зупинкою обчислення при перевищенні заданої кількості ітерацій з видачею інформації для прийняття рішення, що робити далі (продовжити з такою ж кількістю ітерацій;
- виконання програми до кінця, поки не буде знайдено корень рівняння, незалежно від того скільки ітерацій це буде потребувати;
- передбачити вихід із програми, перед виходом вивівши на екран отриманий проміжний результат підрахунку;
- реалізувати визначення затрат часу на пошук кореня, наприклад, за допомогою функцій *clock*() чи *time*();
- передбачити вивід на екран значення кореня та значення функції в цій точці.

7. Розрахувати значення похідної і підтвердити наявність одного рішення рівняння на проміжку. Дослідити збіжність методів від вибору початкових умов.

8. 8. Результати досліджень відобразити у звіті.

Теоретичні відомості ЛР 3. Частина 2

Для розробки програми рішення нелінійного рівняння потрібно ознайомитися з основами теорії рішення нелінійного рівняння, що надано у першій частині ЛР 3. Далі розглянемо наближений метод розв’язку рівняння - **метод хорд**. Нехай функція $f(x)$ на відрізку $[a, b]$ задовольняє таким умовам:

– рівняння $f(x) = 0$ має єдиний розв’язок, точне значення якого позначимо через ξ;

– $f(x)$ є неперервною на [a, b];

– $f'(x)$ зберігає знак на проміжку ізоляції. Якщо $f'(x) > 0$ на відрізку $[a, b]$, то графік кривої $y = f(x)$ буде опуклим униз і можливі два випадки обчислень розв’язку.

Перший випадок: $f(a) > 0$ (рис. 18). У цьому разі за початкове (нульове) наближення візьмемо точку $x_0 = b$.

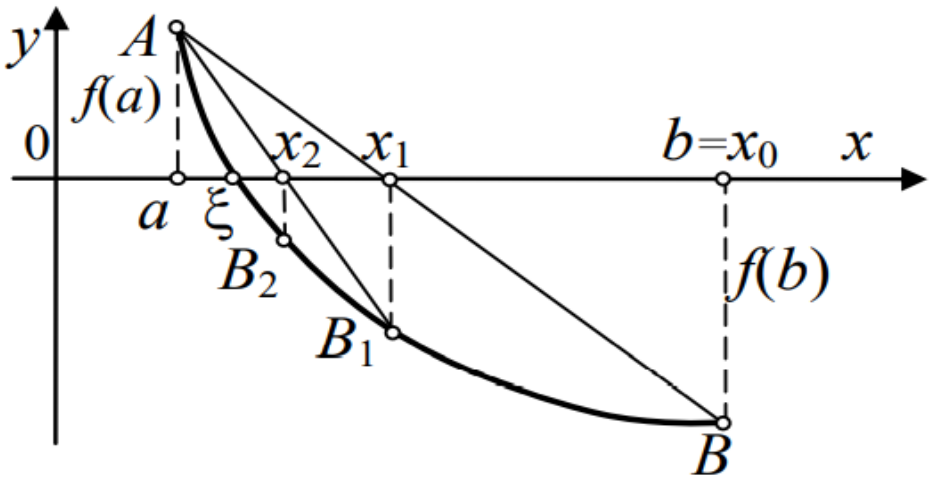


Рис. 18 — Геометрична інтерпретація пошуку розв’язку у разі $f(a) > 0$ [5]

Далі, сполучаючи точки A та B хордою, дістанемо точку x_1. Це і буде перше наближення. Точка x_1 є абсцисою точки $B1$ графіка функції $y = f(x)$. Сполучаючи A та $B1$ хордою, здобудемо друге наближення x_2. Продовжуючи цей процес, обчислимо послідовність точок $x_3, x_4, \ldots, x_n$, що наближаються до розв’язку ξ рівняння $f(x) = 0$.

Для віднайдення x_1 складемо рівняння хорди AB:

$$\frac{y - f(b)}{f(b) - f(a)} = \frac{x - b}{b - a}.$$

Точка x_1 є точкою перетинання хорди із віссю абсцис. Отже, покладемо в рівнянні $b = x_0$, $x = x_1$ і дістанемо:

$$x_1 = x_0 - \frac{f(x_0)}{f(x_0) - f(a)}(x_0 - a)$$

Для обчислення наближення x_2 треба у співвідношенні замінити змінну x_0 на x_1, а змінну x_1 – на x_2:

$$x_2 = x_1 - \frac{f(x_1)}{f(x_1) - f(a)}(x_1 - a).$$

Продовжуючи цей процес, отримаємо наступну розрахункову формулу, що узагальнює процес чисельного знаходження кореня рівняння:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(x_{n-1}) - f(a)}(x_{n-1} - a), \text{ де } x_0 = b, \quad n = 1, 2, \ldots$$

Послідовність чисел $x_0, x_1, \ldots, x_n, \ldots$ є монотонно спадаючою й обмеженою знизу ($x_n > \xi$), тобто при n що прямує до нескінченності значення x_n є наближеним розв'язком рівняння $f(x) = 0$.

Другий випадок: $f(a) < 0$ (рис. 19).

У цьому разі нерухомим є правий кінець відрізка $[a, b]$, і, за аналогією з випадком 1 для послідовних наближень, які утворюють зростаючу послідовність, здобудемо формулу:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(b) - f(x_{n-1})}(b - x_{n-1}), \text{ де } x_0 = a, \quad n = 1, 2, \ldots$$

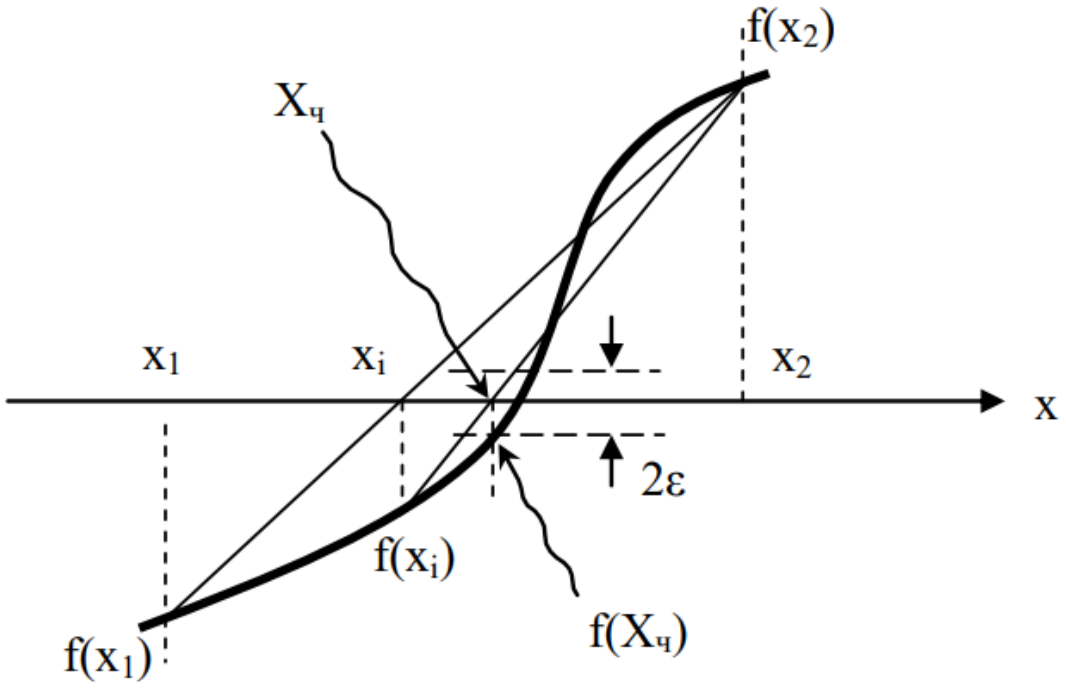


Рис. 19 — Геометрична інтерпретація пошуку розв'язку у разі $f(a) < 0$ [5]

Якщо $f''(x) < 0$ на відрізку $[a, b]$, то слід змінити знак функції на протилежний, тобто розглядати функцію $-f(x)$.

Розглянемо коротко **метод половинного ділення**.

Задамо точки x_1 і x_2, в яких значення функції мають протилежні знаки (в нашому випадку, наприклад, в точці x_1 значення функії $f(x_1)<0$, в точці x_2 значення функії $f(x_2)>0$). Геометрична інтерпретація показан на рис. 20. При цій умові корінь рівняння лежить між точками x_1 *і* x_2. Метод половинного ділення полягає в зменшенні інтервалу $[x_1; x_2]$ вдвічі на кожній ітерації. Обираємо нову точку на середині відрізку $[x_1; x_2]$:

$$x_i=(x_1+x_2)/2$$

Обчислюємо значення $F=f(x_i)$. Перевіряємо, в якому з утворених проміжків $[x_1; x_i]$ та $[x_i; x_2]$ знаходиться розв'язок рівняння (якщо значення функції на кінцях проміжку мають протилежні знаки, то розв'язок знаходиться в цьому проміжку).

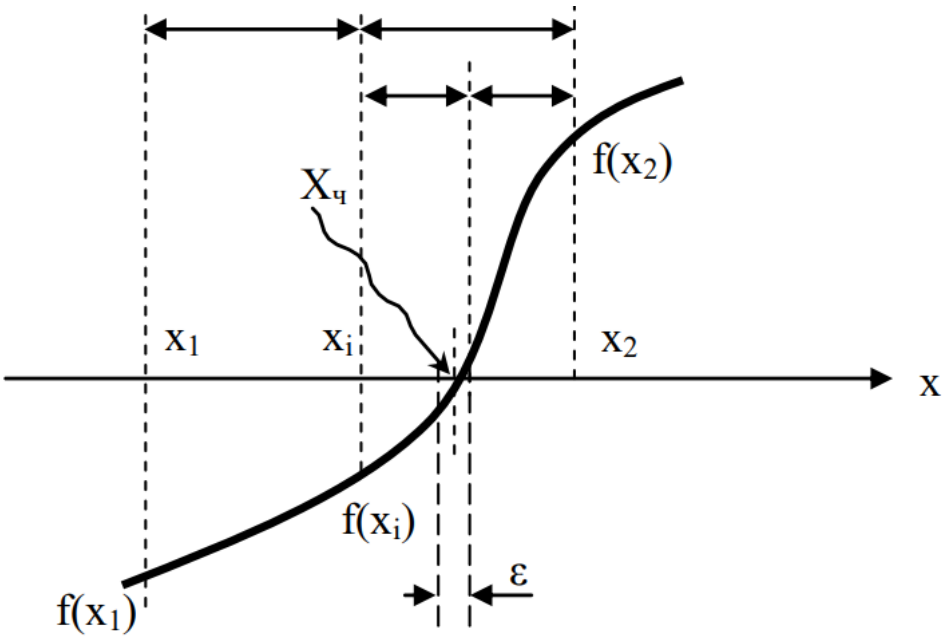


Рис. 20 — Геометрична інтерпретація методу половинного ділення [5]

Для показаної на малюнку функції бачимо, що якщо $F<0$, то корінь рівняння лежить поміж x_1 та x_i. В цьому разі необхідно значення змінної x_1 замінити на значення x_i і повторити обчислення.

Якщо $F>0$, то корінь лежить поміж x_i і x_2, тоді треба замінити значення змінної x_2 на x_i і повторити обчислення. Після присвоєння x_i необхідно перевірити критерій закінчення пошуку кореня нелінійного рівняння.

4.3. Контрольні запитання до лабораторної роботи № 3

1. Яке значення буде повертати наступна функція в ході свого виконання?

```
int func( void )
{
        return 1;
        return 2;
        return 3;
}
```

2. Яке значення буде повертати наступна функція в ході свого виконання?

```
int func( void )
{
        int a = 2;
        int b = 5;
        int c = 12;
        return a, b, c;
}
```

3. Що виконує функція, яка має наступний опис?

```
int func( int x )
{
        return ( x < 0 )  ?  –x : x;
}
```

4. Що виконує функція, яка має наступний опис?

```
int func( int a, int b )
{
        return (a >= b)  ?  a : b;
}
```

5. Що виконує функція, яка має наступний опис?

```
void func( int y)
{
        return;
        y *= y;
        printf("y=%d", y);
}
```

6. Напишіть функцію, яка повертає мінімальне значення серед чотирьох аргументів типу *double*, які передаються в цю функцію.

5. ЛАБОРАТОРНА РОБОТА № 4 «ДИНАМІЧНІ МАСИВИ»

Мета роботи: набуття знань, умінь та навичок з розробки ПЗ з використанням динамічних масивів мови С у процедурній парадигмі.

Лабораторна робота складається з двох частин.

Підготовлені студенти можуть відразу виконувати другу частину лабораторної роботи без зменшення максимальної кількості балів, що нараховується за лабораторну роботу.

Перша частнина передбачає виконання завдання на основі обробки одновимірних масивів з метою набуття навичок, що необхідні для підготовки другої частини лабораторної роботи. Після виконання та захисту першої частини лабораторної роботи вона вважається захищеною у цілому. Максимальна кількість балів, що нараховується за першу частину лабораторної складає половину від максимальної кількості балів за другу частину лабораторну роботу.

Друга частина лабораторної роботи передбачає виконання повного завдання відповідно до силабусу даної дисципліни і передбачає роботу з двовимірними масивами.

За другу частину лабораторної робти, в разі успішного виконання та захисту, нараховується максимальна кількість балів з урахуванням штрафних і додаткових балів.

5.1. Частина перша. Одновимірні динамічні масиви

Вхідні дані ЛР 4. Частина 1

Розмір одновимірного динамічного масиву та значення елементів масиву. Алгоритми обробки одновимірного масиву, що треба реалізувати в першій частині лабораторної роботи № 4.

Вихідні дані ЛР 4. Частина 1

Підраховане середнє значення елементів масиву, знайдене максимальне та мінімальне значення у масиві.

Завдання

Виконання лабораторної роботи передбачає опанування функцій мови програмування С по роботі із динамічною пам'яттю з метою створення одновимірних динамічних масивів. Під час роботи потрібно виконати наступне:

1. Запустити програму, перевірити правильність виконання обрахунків.

2. Написати функцію, що розраховує середнє значення елементів масиву A, що містить елементи типу *unsigned int*. Прототип такої функції може мати вигляд:

```
double mean_value(const unsigned int * A, unsigned int Size );
```

Функція повинна знаходити середнє значення в масиві A, повертати результат, де знайдене середнє значення повинно присвоюватися відповідній змінній на ім'я *average_value*, яку треба оголосити в програмі в функції *main*(). Така змінна може бути оголошена наступним чином:

```
double  average_value;
```

Змінна *average_value* отримує значення, яке повертається функцією *mean_value*(), і потім значення змінної *average_value* повинно бути відображене на екрані (терміналі).

3. Написати функцію, яка знаходить мінімальне значення в масиві A. Прототип такої функції може мати вигляд:

```
unsigned int find_min(const unsigned int * A, unsigned int Size );
```

Функція повинна знаходити мінімальне значення в масиві A, повертати результат, де він повинен присвоюватися відповідній змінній на ім'я *min_value*,

яку треба оголосити в програмі у функції *main*(). Така змінна може бути оголошена наступним чином:

```
unsigned int min_value;
```

Змінна *min_value* отримує значення, яке повертається функцією *find_min*(). Після цього значення змінної *min_value* повинно бути відображене на екрані (терміналі).

4. Написати функцію, яка знаходить максимальне значення в масиві B. Прототип такої функції може мати вигляд:

```
float find_max(const float * B, unsigned int Size );
```

Функція повинна знаходити максимальне значення в масиві B, повертати результат, де він повинен присвоюватися відповідній змінній на ім'я *max_value*, яку треба оголосити в програмі в функції *main*(). Така змінна може бути оголошена наступним чином:

```
float max_value;
```

Змінна *max_value* отримує значення, яке повертається функцією *find_max*(). Після цього значення змінної *max_value*, повинно бути відображене на екрані.

5. Внести зміни у функцію *fill_float_array*(), щоб значення елементів масиву вводились не з клавіатури, а визначалися б за допомогою наступного виразу:

```
B[j] = 5.0 * j + 5.0;
```

6. Внести зміни у функцію *fill_int_array*(), щоб значення елементів масиву вводились не з клавіатури, а визначається за допомогою наступного виразу:

```
A[j] = Size - (j + 1);
```

де *Size* — розмір масиву.

7. Підготувати звіт. Результат розташувати на https://github.com/.

Програма проведення експерименту ЛР 4. Частина 1

Розробка програми передбачає написання коду у текстовому редакторі та компіляцію цього коду у IDE Code::Blocks. Для розуміння процесу створення програми, наведемо у цілому послідовність дій, яку необхідно зробити для створення проекту.

1. Провести первинний аналіз завдання, розробити формалізований опис задачі, обрати узагальнену моделі рішення.

2. Написати код, що реалізує текстовий інтерфейс, у якому передбачено тестування розроблених функцій на мові програмування С відповідно до завдання.

3. Доповнити код п.2, реалізувавши функції відповідно до завдання. Налагодити і протестувати рішення.

4. Виконати налагодження і тестування рішення. Результат розташувати на https://github.com/.

5. Оформити звіт до першої частини лабораторної роботи відповідно до вказаних вимог. Звіт також розташувати на https://github.com/.

Теоретичні відомості ЛР 4. Частина 1

Масиви у мові програмування С (та інших мовах) – це спеціальний механізм для зберігання, упорядкування і обробки великих обсягів даних. Оголошення масиву починається із зазначення його типу даних, потім записується його ім’я та квадратні дужки, в яких вказується розмір масиву, тобто кількість елементів в масиві, наприклад:

```
int array_1 [22];
```

Можна навести інший приклад, де використовується директива препроцесора – макровизначення для задання розміру масиву у вигляді іменованої константи:

```
# define  ARRAY_SIZE  3

...

int array_2 [ARRAY_SIZE];
```

У більшості випадків у квадратних дужках міститься константа, що визначає розмір масиву. Цей масив формується на стеці – спеціалізованій ділянці оперативної пам'яті процесу у операційній системі. Такий масив називають статичним. У цьому випадку потрібно обов’язково вказати розмір масиву до початку виконання програми.

Ініціалізація масиву виконується у різні способи. Один з найбільш простих способів – це ініціалізація статичного масиву у вихідному коді. Вона реалізується за допомогою записаного у фігурних дужках списку елементів масиву. Наприклад:

```
int array_2 [ARRAY_SIZE] = {-1, 0, 1};
```

Крім зберігання числової інформації, масиви також використовуються при роботі із символьними рядками.

Символьні рядки у мові програмування С представляють один з найбільш важливих типів даних. У мові програмування С рядок – це масив типу *char*, але на відміну від звичайного масиву, у символьному рядку обов’язково повинен зберігатися нуль-символ *'\0'*. Він завжди розташовується у кінці символьного масиву і використовується для того, щоб позначити кінець рядка символів. Ініціалізація масивів символьного типу може бути реалізована у різні способи. Найбільш просто вона реалізується за допомогою рядка літералів, записаного у подвійних лапках *"..."*. Такий рядок по-замовчуванню буде містити у кінці символ *'\0'*. Треба відмітити, що його не вказують явно. Приклад ініціалізації,

який показано далі, дещо відрізняється від створення звичайного статичного масиву на стеку, наприклад:

```
char array_3 [ ] = "Hello";
```

Після роботи компілятора у оперативній пам'яті буде розміщений такий рядок:

{ 'H', 'e', 'l', 'l', 'o', '\0' };

Найбільш цікавим елементом мови програмування С, що явно зв'язний з масивами, є вказівник. Він є зручним і ефективним способом доступу до оперативної пам'яті, що відрізняє мову програмування С (С++) від інших високорівневих С-подібних мов. Вказівник робить мову С особливо придатною для системного програмування, розробки операційних систем і програм для мікроконтролерів. Вказівник дає доступ до оперативної пам'яті мікропроцесорної системи, дає можливість маніпулювати адресою і дає доступ до даних. Вказівник не тільки може бути зв'язаний з масивом, він може бути зв'язаний із змінними, що зберігають дані.

Під час оголошення вказівника мають бути присутні два синтаксичних компонента:

- специфікація типу, що задає тип вказівника, який має співпадати з типом змінної, на яку посилається вказівник, якщо передбачається така операція;
- символ зірочка * перед ім’ям вказівника.

Вказівник після оголошення має бути проініціалізований. Для цього він має містити адресу змінної конкретного типу або ж бути зв'язаним із ініціалізованим масивом, або для нього має бути виділена оперативна пам'ять динамічно. Для ініціалізації вказівника і виділення пам'яті динамічно може бути використана спеціальна функція мови програмування С, що описана далі. Ця функція визначає обсяг пам’яті, виділеної вказівнику даного типу. При

використанні функції вважається, що відбувається динамічне створення масиву. При цьому, до завершення програми треба обов'язково звільнити пам'ять особливою функцію мови С, для всіх вказівників, що реалізовані динамічно.

Вказівники можна ініціалізувати шляхом зв'язування із змінною або масивом при оголошенні. Для цього перед ім’ям змінної використовується операція визначення адреси – &, наприклад:

```
int var;                    // Змінна типу int
int * ptr_var = &var;       // Вказівник на змінну типу int.
```

Для масиву не треба використовувати знак амперсанду. У цьому випадку вказіваник зв'язується за адресою з нульовим елементом масиву. Для доступу до значення вказівника виконується операція, що називається операцією розіменування. Ця операція позначається зірочкою, що записана перед ім’ям вказівника, який вже оголошений і ініціалізований. Саме у такий спосіб виконується операція звернення до вмісту оперативної пам’яті за адресою через вказівник. Під час маніпуляції з адресою через механізм вказівників мови С можна виконувати наступні операції:

- присвоєння певної адреси у різник вказівниках;
- арифметичні операції додавання та віднімання;
- унарні операції інкременту чи декременту;
- порівняння (для вказівників одного типу) у логічних операціях з вказівниками.

Для того, щоб виділити необхідний об’єм динамічної пам’яті для збереження значень елементів одновимірного динамічного масиву, можна скористатися функціями, прототипи яких мають вигляд:

```
void * calloc ( size_t num_Of_Items, size_t size );
void * malloc ( size_t size );
```

Функція *calloc*() повинна отримувати два параметри – кількість елементів в динамічному масиві (параметр *num_Of_Items*), а також розмір в байтах типу даних масиву (іншими словами – необхідно задати скільки байтів займає значення одного елементу масиву обраного типу даних в пам'яті комп'ютера). Для того щоб визначити розмір в байтах обраного типу даних потрібно скористатися операцією *sizeof*.

Особливість динамічних масивів полягає в тому, що розмір масиву задається на етапі виконання програми, а не на етапі її написання як це відбувається при роботі із статичними масивами, для яких розмір задається у вигляді константи. Функція *malloc*() повинна отримувати в якості параметру кількість байтів, яку необхідно виділити в динамічній пам'яті для зберігання значень елементів масиву. Ця кількість байтів може бути розрахована як добуток кількості елементів масиву на кількість байтів, яка виділяється для зберігання значення обраного типу даних.

Наприклад, розмір динамічного масиву задається змінною *Size*. Значення змінної зчитується з клавіатури. Тоді наступний фрагмент коду дозволяє виділити пам'ять для двох динамічних одновимірних масивів, що мають імена A та B, які зберігають значення типу *unsigned int* та *float*. Для виділення пам'яті цим динамічним масивам використовуються функції відповідно *calloc*() та *malloc*():

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    unsigned int * A;
    float * B;
    unsigned int Size;
```

```
    printf("Enter size of array: ");
    scanf("%u", &Size);

    A = (unsigned int *) calloc (Size, sizeof ( unsigned int ) );

    B = (float *) malloc (Size * sizeof ( float ) );

    return 0;
}
```

Враховуючи, що функції *malloc*() і *calloc*() повертають адресу на перший байт пам'яті виділеної області, для вичерпного запису цієї операції в наведеному прикладі також присутня операція приведення типу даних.

При виділенні пам'яті для динамічного масиву A, повернене функцією *calloc*() значення вказівника приводиться до типу даних *unsigned int* *. При виділенні пам'яті для динамічного масиву B, повернене функцією *malloc*() значення вказівника приводиться до типу даних *float* *.

Якщо функції *malloc*() та *calloc*() не змогли виділити пам'ять, вони повертають значення *NULL*. Таким чином, перед тим як продовжувати роботу із динамічним масивом, потрібно перевірити, чи була для нього виділена пам'ять. Якщо пам'ять не була виділена, то в цьому випадку можливо завершити програму за допомогою функції *exit*().

Для того, щоб використовувати функції *exit*(), *malloc*() та *calloc*() необхідно підключити заголовочний файл <*stdlib.h*>. Після додавання перевірки на предмет виділення пам'яті в код попереднього прикладу, отримаємо:

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main()
{
    unsigned int * A;
    float * B;
    unsigned int Size;

    printf("Enter size of array: ");
    scanf("%u", &Size);

    A = (unsigned int *) calloc (Size, sizeof ( unsigned int ) );

    B = (float *) malloc (Size * sizeof ( float ) );

    if( A == NULL  ||  B == NULL ){
        printf("Memory has not been allocated");
        exit(0);
    }

    return 0;
}
```

Після того, як відбулося успішне виділення пам'яті для динамічних масивів, робота із ними може виконуватися аналогічно як із статичними масивами. Наприклад, для того щоб задати значення для елементів масиву A з клавіатури, може бути написана окрема функція, що буде мати ім'я *fill_int_array*(), і яка може мати наступний прототип:

```
void  fill_int_array ( unsigned int *  uiptr,  unsigned int  size_of_array );
```

Імена параметрів в прототипах ігноруються компілятором, але для того, щоб дати більше інформації про призначення відповідних параметрів, вони були вказані в прототипі, при цьому імена параметрів були обрані такими, щоб дати користувачу зрозуміти призначення відповідних параметрів.

Інша функція, ім'я якої *fill_float_array*(), призначена для присвоєння елементам масиву B відповідних значень. Прототип функції має вигляд:

```
void  fill_float_array ( float *  fptr, unsigned int  size_of_array );
```

Також необхідно написати дві функції, які будуть виводити на екран значення елементів масивів A та B, які зберігають значення елементів, що мають тип даних *unsigned int* та *float* відповідно. Враховуючи, що дані функції не передбачають зміну значень елементів масивів, а лише виконують виведення їх значень на екран, то можна використати службове слово *const* щоб унеможливити модифікацію значень елементів масивів у відповідних функціях. Прототипи таких функцій будуть наступними:

```
void  print_int_array ( const unsigned int *  uiptr,  unsigned int  size_of_array );

void  print_float_array ( const float *  fptr,  unsigned int  size_of_array );
```

Також, необхідно написати функції, які б виконували певну обробку масивів. Наприклад, нехай перша функція знаходить суму елементів в масиві A (елементи масиву мають тип даних *unsigned int*) , а друга функція знаходить добуток елементів масиву B (елементи масиву мають тип даних *float*). Оскільки передбачається, що функції не будуть змінювати значення елементів масивів, тому також можна використати службове слово *const*, щоб відповідні функції сприймали масив як константний, тобто такий, який не можна модифікувати. Кожна окрема функція буде отримувати вказівник на перший елемент масиву та розмір масиву, а повертати буде відповідно суму і добуток елементів масиву. Прототипи таких функцій можуть мати наступний вигляд:

```
unsigned int  sum ( const  unsigned int *  uiptr, unsigned int  size_of_array );
```

```
float  product ( const float *  fptr,  unsigned int  size_of_array );
```

Функція *sum*() буде знаходити суму елементів масиву A, а враховуючи, що елементи мають тип даних *unsigned int*, тому і результат, який повертає функція *sum*() буде мати тип даних *unsigned int*.

Аналогічно, функція *product*() буде знаходити добуток елементів масиву B, а враховуючи, що елементи масиву мають тип даних *float*, тому і результат, який буде повертатися функцією повинен мати відповідний тип даних. Але враховуючи, що добуток може бути досить великим по величині (або дуже малим), то щоб коректно зберегти результат і повернути його в місце виклику функції *product*(), то замість типу даних *float* (тип даних результату, що повертає функція *product*()), можна було б обрати тип даних *double* для забезпечення кращої точності результату обрахунку. Тоді прототип функції *product*() буде мати вигляд:

```
double  product ( const float *   fptr,  unsigned int  size_of_array );
```

Після того, як всі дії над масивами було виконано, необхідно звільнити пам’ять, яка раніше була виділена для масивів A та B. Це можна зробити наступним чином:

```
free ( A );

free ( B );
```

Текст програми, в якому виконуються обумовлені дії, може мати наступний вигляд:

```
#include <stdio.h>
#include <stdlib.h>

//-------------------------------   Прототипи   ------------------------------------

void fill_int_array( unsigned int * uiptr, unsigned int size_of_array );
```

```
void fill_float_array( float * fptr, unsigned int size_of_array );
void print_int_array( const unsigned int * uiptr, unsigned int size_of_array );
void print_float_array( const float * fptr, unsigned int size_of_array);

unsigned int sum ( const  unsigned int * uiptr, unsigned int size_of_array );
double product( const float * fptr, unsigned int size_of_array );

//----------------------------   Опис функції main()   ---------------------------------------
int main()
{
   unsigned int * A;
   float * B;

   unsigned int Size;   // змінна для збереження розміру динамічного масиву
   unsigned int amount; // змінна для збереження суми елементів масиву A
   double mult;         // змінна для збереження добутку елементів масиву B

   printf("Enter size of array: ");
   scanf("%u", &Size);

   A = (unsigned int *) calloc (Size, sizeof(unsigned int) );
   B = (float *) malloc (Size * sizeof(float) );

   if( A == NULL  ||  B == NULL ){
      printf("Memory has not been allocated");
      exit(0);
   }
```

```
    fill_int_array( A, Size );
    fill_float_array( B, Size );
    print_int_array( A, Size );
    print_float_array( B, Size );

    amount = sum( A, Size );
    mult = product( B, Size );
    printf("\n\nSum = %u", amount);
    printf("\nProduct = %.3lf", mult);

    free(A);
    free(B);
    return 0;
}
//--------------------------------------  Опис Функцій  --------------------------------------
void fill_int_array( unsigned int * A, unsigned int Size )
{
     unsigned int j;
     unsigned int temp;
     printf("\n\nEnter values of UNSIGNED INT elements of array.\n");
     for( j = 0;  j <= Size-1;  j++ ){
         printf("A[%u]= ", j);
         scanf("%u", &temp);
         A[j] = temp;
     }
}
```

```
//-------------------------------------------------------------------------------------------------
void fill_float_array( float * B, unsigned int Size )
{
    unsigned int j;
    float temp;

    printf("\n\nEnter values of FLOAT elements of array.\n");

    for( j = 0;  j <= Size-1;  j++ ){
        printf("B[%u]= ", j);
        scanf("%f", &temp);
        B[j] = temp;
    }
}

//-------------------------------------------------------------------------------------------------

void print_int_array( const unsigned int * A, unsigned int Size )
{
    unsigned int j;

    printf("\n\nArray of UNSIGNED INT values:\n");

    for( j = 0;  j < Size;  j++ )
      printf("%5d", A[j] );
}
```

```
//-------------------------------------------------------------------------------------------
void print_float_array( const float * B, unsigned int Size )
{
    unsigned int j;

    printf("\n\nArray of FLOAT values:\n");

    for( j = 0;  j < Size;  j++ )
      printf("%7.2f", B[j] );
}

//-------------------------------------------------------------------------------------------

unsigned int sum (const  unsigned int * A, unsigned int Size )
{
    unsigned int j, S;

    S = 0;     // змінна, яка використовується для обрахунку значення
               // суми елементів масиву

    for( j = 0;  j < Size;  j++ )
      S += A[j];

    return S;
}
```

```
//-------------------------------------------------------------------------------------------
double product(const float * B, unsigned int Size )
{
    unsigned int j;
    double M;

    M = 1.0;   //змінна, яка використовується для обрахунку значення
               // добутку елементів масиву

    for( j = 0;  j < Size;  j++ )
      M *= B[j];

    return M;
}
```

5.2. Частина друга. Двовимірні динамічні масиви

Вхідні дані ЛР 4. Частина 2

Дві матриці (A, B). Матриця A є квадратною, B – прямокутною. Матриці задаються у вигляді динамічних масивів.

Розмірність масиву A становить n_a, де n_a – номер варіанту.

Розмірність масиву B становить n_b, m_b, де m_b обирається довільно з умовою виконання завдання.

Значення елементів масиву, можуть вводитися з клавіатури або можуть задаватися у циклі з використанням генератору випадкових чисел, або формуватися у відповідності до певного довільного обраного математичного виразу, що дозволяє розрахувати поточне значення елементів масиву на основі

значення попереднього елементу масиву, або ж на основі значення індексу елементу. Передбачити вказані варіанти формування значення елементів масиву за вибором користувача. Допускається формування елементів масиву за простими формулами (наприклад, $A_{i,j} = i+j$).

Вихідні дані ЛР 4. Частина 2

Максимальний елемент матриці та мінімальний з елементів, що знаходяться нижче (вище) головної діагоналі. Результат транспонування матриці. Результат матричного добутку. Результати сортування матриці. Програма має бути реалізована у процедурній парадигмі, з функціями мови програмування С.

Завдання

Виконання лабораторної роботи передбачає використання функцій по роботі із динамічною пам'яттю з метою створення двовимірних динамічних масивів.

Реалізація і прототипи функцій — рішень завдань, що надані далі, можуть бути змінені за рішенням розробників. При цьому вимагається передавати масиви у функції в якості параметрів, користуючись програмним механізмом вказівників мови С.

Використовувати глобальні змінні і статичні масиви не дозволяється!

Перед початком обробки вивести на екран масив, що обробляється. Перед завершенням програми звільнити виділену під масиви пам'ять.

1. Знайти максимальний та мінімальний елемент двовимірної матриці A, що знаходяться для парних варіантів – нижче головної діагоналі, а для непарних варіантів – вище головної діагоналі. Написати функцію, яка вирішує це завдання для масиву A. Прототип такої функції може мати вигляд:

```
void max_min ( int ** A, int n_a, int * max_a , int * min_a );
```

2. Транспонувати двовимірну матрицю B. Написати функцію, яка знаходить результат транспонування. Прототип такої функції може мати вигляд:

```
void  transpose_b ( int *** B, int  n_b,  int  m_b );
```

або

```
int **  transpose_b ( int ** B,  int  n_b,  int  m_b );
```

3. Знайти матричний добуток A×B. Результат зберегти в матриці C. Написати функцію, яка знаходить A×B. Прототип такої функції може мати вигляд:

```
int **  mult_a_b  ( int ** B,  int  n_b,  int  m_b,  int ** A,  int  n_a );
```

4. Відсортувати за зростанням елементи масиву A, а також передбачити можливість сорутвання елементів лише в межах обраного рядка матриці A. Написати функцію, яка виконує сортування.

5. Вивести на екран суму елементів рядків матриці A та стовпців матриці B. Написати функцію, яка знаходить результат.

6. Після виконання вибраного пункту передбачити можливість продовження роботи програми.

7. Реалізувати відповідний текстовий інтерфейс користувача.

8. Оформити звіт до лабораторної роботи відповідно до вимог. Звіт і вихідний код розташувати на https://github.com/. Побудувати блок-схему алгоритму однієї з функцій програми.

Програма проведення експерименту ЛР 4. Частина 2

Розробка програми передбачає написання коду у текстовому редакторі та компіляцію цього коду у IDE Code::Blocks. Для розуміння процесу створення програми, наведемо у цілому послідовність дій, яку необхідно зробити для

створення проекту. Допускаються інші варіанти середовища розробки за вибором студента.

1. Провести первинний аналіз завдання, розробити формалізований опис задачі, обрати узагальнену моделі рішення.

2. Написати код, що реалізує функції, які дозволяють зробити виділення пам’яті та виконати ініціалізацію двовимірних масивів А і В. Протестувати функції.

3. Написати код, що реалізує функцію, яка знаходить мінімальне значення в масиві А. Протестувати функцію на прикладі матриць п.2.

4. Написати код, що реалізує функцію, яка робить транспонування двовимірної матриці Б.

5. Написати код, що реалізує матричний добуток А×В. Виконати перевірку правильності роботи функції, яка реалізує знаходження матричного добутку.

6. Написати функції, що роблять всі інші пункти завдання, у тому числі вивід результатів на екран. Протестувати рішення.

7. Реалізувати текстовий інтерфейс користувача.

8. Оформити звіт до лабораторної роботи відповідно до наведених вимог. Звіт і вихідний код розташувати на https://github.com/. Побудувати блок схему алгоритму однієї з функцій програми.

Математичне супроводження операцій над двовимірними масивами

У мові програмування С (С++) можна реалізувати двовимірні масиви. Математична модель двовимірних масивів у мові програмування С (С++) може бути з певними обмеженнями розглядатися як така, що відповідає математичному терміну «матриця». Матрицею у математиці називають прямокутну таблицю з чисел, яка містить деяку кількість m рядків і деяку кількість n стовпців, що чітко упорядковані. Отже, двовимірному масиву

відповідає математичне поняття двовимірної матриці, приклад оголошення двовимірного масиву показаний далі:

```
int   arr [ 3 ][ 4 ];
```

Повернемося до математичного визначення, що може бути таким: матриця це сукупність чисел, записаних у вигляді таблиці із *m* рядків і *n* стовпців. Доступ до значень елементів масиву відбувається через вказання індексів елементу.

У математиці натуральні числа *m* та *n* задають розмірність матриці, яку позначають як *m*x*n*. Тобто, якщо говорять про те, що задано матрицю розмірністю, наприклад, 3x5, то це матриця з трьома рядками і п'ятьма стовпцями. Матрицю розмірністю *m*x*n* можна позначити M_{mxn}. Відповідно інші матриці позначаються великими латинськими літерами (A, B, C, …), елементи матриці – маленькими літерами. Слід зауважити, що перше число завжди позначає кількість рядків *m*.

Якщо в матриці однакова кількість рядків та стовпців, то таку матрицю називають квадратною і кажуть, що задано матрицю *n*-го (або *m*-го) порядку.

Аналогом такої квадратної матриці буде, наприклад, масив:

```
#define  ARR_SIZE  5

int   arr [ ARR_SIZE ] [ ARR_SIZE ];
```

Для позначення місця елементу у матриці використовується індексний запис a_{ij} . Цей елемент знаходиться на перетині *i*-го рядка та *j*-го стовпця матриці A. Діапазон для *i*-го рядка та *j*-го стовпця у математиці визначають у наступний спосіб:

$$i = 1, 2, \ldots m; \quad j = 1, 2, \ldots, n.$$

У програмуванні індекси для рядків і стовпчиків масиву прийнято також позначати такими самими буквами латинського алфавіту. Покажемо загально

прийнятний програмний приклад ініціалізації масиву на ім'я *arr*, що відповідає матриці M_{mxn}, яка згадувалася вище. Ініціалізація відбувається нулями:

```
for ( i=0;  i < ARR_SIZE;  i++)

    for ( j=0;  j < ARR_SIZE;  j++)

        arr [ i ] [ j ] = 0;
```

У математиці також прийнятий запис для матриці у вигляді таблиці (елементи в круглих дужках або у подвоєних рисках) [7]

$$\begin{pmatrix} a_{11} & a_{12} & .... & a_{1n} \\ a_{21} & a_{22} & .... & a_{2n} \\ .... & .... & .... & .... \\ a_{m1} & a_{m2} & .... & a_{mn} \end{pmatrix} \quad \text{або} \quad \begin{Vmatrix} a_{11} & a_{12} & .... & a_{1n} \\ a_{21} & a_{22} & .... & a_{2n} \\ .... & .... & .... & .... \\ a_{m1} & a_{m2} & .... & a_{mn} \end{Vmatrix}.$$

У випадку квадратної матриці n-го порядку:

$$\begin{pmatrix} a_{11} & .... & a_{1n} \\ .... & .... & .... \\ a_{n1} & .... & a_{nn} \end{pmatrix}.$$

Корисним буде ще декілька математичних визначень, що наведені у [7]:

- **головна діагональ** – діагональ, що починається з лівого верхнього кута матриці та прямує в правий нижній кут (елементи $a_{11}, a_{22}, ..., a_{nn}$);

- **побічна діагональ** – діагональ, що починається з лівого нижнього кута матриці та прямує в правий верхній кут (елементи $a_{n1}, a_{n-1,2}, ..., a_{1n}$);

- **слід матриці** (позначається Tr A або Sp A) – сума елементів головної діагоналі.

Звичайно, що мова програмування дає можливість доступу до головної діагоналі, побічної діагоналі, можна розрахувати слід матриці.

Індекси елементів матриці є парами натуральних чисел (*i, j*) [7]. Тобто, вони є елементами декартового добутку множин $I \times J = \{(1,1),(1,2),...,(i,j),...,(m,n)\}$.

Фактично, кожній парі чисел декартового добутку $I \times J$ матриця ставить у відповідність елемент матриці a_{ij} – дійсне чи комплексне число. Таким чином, матриця є зображенням декартового добутку $I \times J$ на множину дійсних чисел R (якщо елементи матриці є дійсними числами – матриця називається дійсною).

У іншому випадку матриця є зображенням на множину комплексних чисел [7]. У мові програмування С не передбачений тип даних комплексних чисел. Програмно його можна реалізувати або застосувати використовуючи спеціальні бібліотеки. Аналогом одновимірного масиву у математиці, є випадок коли одна із розмірностей матриці дорівнює одиниці:

$$\mathrm{B} = \begin{pmatrix} b_{11} \\ b_{21} \\ ... \\ b_{m1} \end{pmatrix}$$

Матриці такого типу також називаються векторами (вектор з алгебраїчної точки зору) – вектор-рядок і вектор-стовпець відповідно. Матриця 1-го порядку, де *m=n=1,* це звичайне число. Корисними різновидами матриці для роботи програміста є наступні:

- **нульова матриця** (нуль-матриця) – матриця розмірності $m \times n$, де всі елементи такої матриці дорівнюють нулю:

$$\mathrm{O} = \begin{pmatrix} 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & 0 \end{pmatrix};$$

- **діагональна матриця** – квадратна матриця *n*-го порядку, в якій всі елементи крім елементів головної діагоналі дорівнюють нулю:

$$\mathrm{D}=\begin{pmatrix} a_{11} & 0 & \ldots & 0 \\ 0 & a_{22} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & a_{nn} \end{pmatrix}=\mathrm{diag}\left(a_{11},\quad a_{22},\quad \ldots,\quad a_{nn}\right);$$

- **одинична матриця** – це діагональна матриця, в якій всі елементи головної діагоналі дорівнюють одиниці

$$\mathrm{I}=\begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 \end{pmatrix};$$

- **трикутна матриця** – квадратна матриця *n*-го порядку, в якій всі елементи під/над головною/побічною діагоналлю нульові, наприклад:

а) верхня трикутна матриця відносно головної діагоналі:

$$\begin{pmatrix} a_{11} & a_{12} & \ldots. & a_{1n} \\ 0 & a_{22} & \ldots. & a_{2n} \\ \ldots. & \ldots. & \ldots. & \ldots. \\ 0 & 0 & \ldots. & a_{mn} \end{pmatrix}$$

б) нижня трикутна матриця відносно побічної діагоналі:

$$\begin{pmatrix} 0 & \ldots. & 0 & a_{1n} \\ 0 & \ldots. & a_{2,n-1} & a_{2n} \\ \ldots. & \ldots. & \ldots. & \ldots. \\ a_{m1} & \ldots. & a_{m,n-1} & a_{mn} \end{pmatrix}$$

Основні операції, які можна виконувати над матрицями у математиці можуть бути реалізовані програмно через двовимірні або одновимірні масиви.

Наприклад, математичне порівняння матриць у контексті чи дорівнює одна матриця іншій, це випадок коли дві матриці A та B перевіряють на рівність. Якщо ці матриці мають однакову розмірність та всі їх відповідні елементи співпадають, то ці матриці однакові. Відповідні перевірки можна реалізувати програмно. Математичний приклад, запозичений з [7]:

$$A = \begin{pmatrix} 1 & -4 & -3 \\ 0 & 2 & 5 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 \\ -4 & 2 \\ -3 & 5 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 7 & -3 \\ 0 & 2 & -5 \end{pmatrix}, \quad D = \begin{pmatrix} 1 & -4 & -3 \\ 0 & 2 & 5 \end{pmatrix},$$

$A \neq B$ (різна розмірність),

$A \neq C$ (різні елементи),

$A = D$.

Програмно також можна виконати суму матриць, різницю матриць, розрахунок оберненої матриці, множення матриці на число. Докладно про ці операції над матрицями можна прочитати у [7].

Важливими властивостями лінійних операцій над матрицями, що є корисними при розробці відповідних алгоритмів, і часто можуть застосовуються на практиці при розробці програм, де виникає необхідність обробки даних, які зазвичай представляються у формі матриць, є наступні [7]:

- комутативність додавання (розподільна властивість):

$$A + B = B + A,$$

- асоціативність додавання (сполучна властивість):

$$(A + B) + C = A + (B + C),$$

- додавання нульової матриці:

$$A + 0 = A = 0 + A,$$

- сполучна властивість відносно числового множника:

$$\alpha(\beta A)=(\alpha\beta)A,\quad \alpha,\beta\in R$$

- розподільна властивість відносно суми матриць:

$$\alpha(A+B)=\alpha A+\alpha B,$$

- розподільна властивість відносно суми чисел:

$$(\alpha+\beta)A=\alpha A+\beta A,$$

Транспонуванням матриці називають таку операцію, коли міняють місцями рядки та стовпці матриці із збереженням порядку їх слідування. Матрицю, транспоновану відносно матриці А, позначають A^T (або A')

Якщо

$$A=\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{pmatrix},$$

то

$$A^T=\begin{pmatrix} a_{11} & \dots & a_{m1} \\ \dots & \dots & \dots \\ a_{1n} & \dots & a_{mn} \end{pmatrix},$$

Приклад [7]:

$$A=\begin{pmatrix} 1 & -1 & -4 \\ -2 & 3 & 5 \\ 7 & 8 & -9 \end{pmatrix},\qquad A^T=\begin{pmatrix} 1 & -2 & 7 \\ -1 & 3 & 8 \\ -4 & 5 & -9 \end{pmatrix};$$

$$B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \qquad B^T = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}.$$

Властивості операції транспонування[7]:

1. $\left(A^T\right)^T = A,$

2. $\left(\alpha A\right)^T = \alpha A^T,$

3. $\left(A + B\right)^T = A^T + B^T.$

Наведемо функції мови програмування С, що є навчальними програмними прикладами для реалізації математичних операцій з матрицями.

Перша функція виділяє динамічну пам'ять для двовимірного масиву, що відповідає двовимірній матриці і повертає вказівник на двовимірний масив, що відповідає математичній моделі двовимірної матриці.

```
int  ** memory_allocation ( int  rows, int  cols )
{
   int  i = 0;
   int ** matrix = ( int ** ) malloc ( rows * sizeof ( int * ) );
   for( i = 0; i < rows; i++)
   {
     matrix[ i ] = ( int * ) malloc ( cols * sizeof( int ) );
   }
   return matrix;
}
```

Друга функція призначена для звільнення пам'яті, що виділена динамічно:

```
void  clearMemory ( int ** matrix,  int rows )
{
   int i;
   for( i = 0;  i < rows;  i++ )
   {
      free( matrix[ i ] );
   }
   free( matrix );
}
```

Третя функція показує можливість роботи з двовимірними масивами. Даний приклад показує пошук максимального значення у квадратній матриці, що містить додатні елементи.

```
int  Maximum ( int ** A, int  N )
{
    int i, j, max;
    max = A[ 0 ] [ 0 ];
    for( i=0; i<N; i++)
   {
          for( j=0; j<N; j++ )
          {
             if ( A[ i ] [ j ] > max )
                   max = A [ i] [ j ];
          }
     }
   return max;
}
```

Четверта функція відображає матрицю в транспонованому вигляді, однак дана функція не транспонує саму матрицю. Тобто, вона не змінює розміщення елементів. Це приклад ***неправильного !!!*** виконання пункту завдання що стосується траспонування матриці:

```
void Transpose ( int ** B, int N, int M )
{
   int  i,j;
   printf("\n transposed matrix B :  \n");
   for ( i=0; i<M; i++ )
   {
      for ( j=0; j<N ;j++ )
      {
         printf("%d \t", B[j][i]);
      }
   printf("\n");
   }
}
```

Важливою операцією є добуток матриць. Перед виконанням добутку матриць потрібно визначитися з терміном узгодження матриць. Матриці $A_{m\times n}$ та $B_{p\times q}$ називаються узгодженими, якщо кількість стовпців n першої матриці дорівнює кількості рядків p другої матриці. Слід зазначити, що лише узгоджені матриці можна множити. Результатом множення матриці $A_{m\times n}$ на матрицю $B_{n\times q}$ є матриця C розмірності $m\times q$, елементи якої визначають таким чином [7]:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + ... + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

Тобто, елементи рядка першої матриці множаться на відповідні елементи стовпця другої матриці та обчислюється сума всіх цих добутків.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{in} \\ - & - & - & \rightarrow \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & \dots & b_{1j} & | & b_{1q} \\ b_{21} & \dots & b_{2j} & | & b_{2q} \\ \dots & \dots & \dots & | & \dots \\ b_{n1} & \dots & b_{nj} & \downarrow & b_{nq} \end{pmatrix} = \begin{pmatrix} \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & c_{ij} & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix}$$

Наприклад, нехай задано матирці:

$$\mathrm{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \text{ та } \mathrm{B} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix},$$

тоді елементи матриці $\mathrm{C} = \mathrm{A} \times \mathrm{B}$ обчислюються наступним чином:

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22} \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} \\ c_{22} &= a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

Властивості добутку матриць [7]:

- **асоціативність** (сполучна властивість):

$$(\mathrm{AB})\mathrm{C} = \mathrm{A}(\mathrm{BC})$$

- **розподільна властивість** відносно суми матриць:

$$\begin{aligned} (\mathrm{A} + \mathrm{B})\mathrm{C} &= \mathrm{AC} + \mathrm{BC}, \\ \mathrm{A}(\mathrm{B} + \mathrm{C}) &= \mathrm{AB} + \mathrm{AC}. \end{aligned}$$

Нульовий степінь квадратної матриці – це одинична матриця $\mathrm{A}^0 = \mathrm{I}$ [7].

Перший степінь квадратної матриці – це сама матриця A [7]:

$$\mathrm{A}^1 = \mathrm{A}$$

5.3. Контрольні запитання до лабораторної роботи № 4

1. Що таке матриця?
2. Які існують види матриць?
3. Як позначають матриці?
4. Які операції можна проводити над матрицями?
5. Які матриці називаються узгодженими?
6. Як перемножити дві матриці?
7. Наведіть визначення масиву та його основні характеристики. У чому полягає особливість розташування масивів у оперативній пам'яті?
8. Як оголосити масив та виконати його ініціалізацію? Наведіть програмні приклади.
9. Як відбувається ініціалізація масивів символьного типу? Наведіть програмні приклади.
10. Опишіть програмний механізм вказівника. Наведіть програмні приклади.
11. Які операції можна виконувати над вказівниками?
12. Як пов’язані між собою масиви і вказівники у мові програмування С.
13. Чим відрізняються статичні і динамічні масиви у контексті їх використання в мові програмування С?
14. Чим відрізняється розташування статичних і динамічних масивів у оперативній пам'яті?

6. ЛАБОРАТОРНА РОБОТА № 5 «ПРОГРАМНІ ПОТОКИ ДЛЯ РОБОТИ З ФАЙЛАМИ»

Мета роботи: отримати навички роботи з текстовими та двійковими (бінарними) файлами при написанні програм на мові програмування С, а також ознайомитися із форматним та не форматним вводом-виводом.

Вхідні дані ЛР 5

Текстовий файл. У вхідному текстовому файлі в кожному окремому рядку знаходиться наступна інформація:

− початкове значення аргументу математичної функції;

− кінцеве значення аргументу математичної функції;

– кількість точок, в яких обчислюється значення математичної функції;

– крок зміни аргументу математичної функції;

– шифр навчальної групи;

– прізвище та ім’я студента, який виконує лабораторну роботу.

Математичний вираз функції обирається з лабораторної роботи 1 або лабораторної роботи 2 відповідно до варіанту. Для збереження значення аргументу (параметр *x*) та значення функції (*f*(*x*)) — обрати тип даних *double*.

Вихідні дані ЛР 5

Проект IDE Code::Blocks, вихідний код програми мовою С. Текстовий файл з результатами виконання відповідно до завдання, бінарний файл із збереженими в ньому даними в ході виконання програми.

Завдання

1. В текстовому редакторі «Блокнот» створити текстовий файл і записати до нього інформацію, що надана у пункті *Вхідні дані ЛР 5*. Обрати допустиме

ім’я файлу та зберегти його в каталозі, де розміщується проект, в якому реалізується завдання лабораторної роботи.

2. Написати програму, яка відкриває текстовий файл, що був створений в п.1. При цьому файл відкривається в режимі читання. Програма зчитує числову величину початкового значення аргументу, яка записана у файлі, і зберігає її в змінну. Аналогічно, зчитати всі інші значення з вхідного текстового.

3. Скористатися результатами лабораторної роботи 1 і виконати побудову таблиці (аналогічної ЛР 1) з використанням даних, що зчитані з файлу п.2.

4. Створити текстовий файл і записати в текстовий файл таблицю п.3. Після таблиці з нового рядка записати шифр групи (шифр групи повинен бути попередньо прочитаний із вхідного текстового файлу і збережений в програмі, наприклад, із використанням символьного рядка або масиву символів, а також аналогічним чином зчитати з вхідного файлу і записати у вихідний файл ім’я та прізвище виконавця лабораторної роботи).

5. Створити двійковий (бінарний) файл і записати в цей файл кількість рядків таблиці, а потім після даного значення записати послідовність значень - «аргумент» та «значення функції». Тобто, бінарний файл містить послідовність даних, які відповідають значенню аргументу та значенню фукнції при заданому аргументі, – така послідовність значень отримується для кожного рядка таблиці і така пара чисел послідовно записується в двійковий (бінарний) файл.

6. Перевірити результати роботи програми. Пересвідчитися, що були створені бінарний та текстовий файли. Переглянути вміст текстового файлу. Оцінити розмір бінарного файлу.

7. Відкрити бінарний файл в програмі, що реалізує ЛР 5. Зчитати перше значення - кількість рядків таблиці (нехай цей параметр, позначається як N). Створити динамічний масив розміром $2N$ для збереження значень типу даних *double*. Зберегти в цей масив набір пар даних «аргумент»-«значення функції» (іншими словами потрібно записати в масив послідовність набору даних x та $f(x)$), що збережені в даному бінарному файлі. Відобразити значення, що були

збережені в динамічному масиві на екрані монітору у вигляді таблиці, що складається із трьох стовпбці — «номер точки» (це фактично номер рядка таблиці), «значення аргументу» (значення параметру *x*), «значення функції» (значення *f*(*x*)). Перевірити, чи вдалося коректно прочитати дані із бінароного файлу і відобразити їх на екрані.

8. Оформити звіт до лабораторної роботи відповідно до вказаних вимог. Звіт і вихідний код розташувати на https://github.com/.

Програма проведення експерименту ЛР 5

На першому етапі роботи треба створити проект і розташувати у каталозі проекту текстовий файл і записати до нього інформацію, що вказана у пункті *Вхідні дані ЛР 5*. Це можна зробити з використанням будь якого текстового редактора. Розробка програми передбачає написання коду у текстовому редакторі та компіляцію цього коду у IDE Code::Blocks. Для розуміння процесу створення програми, наведемо у цілому послідовність дій, яку необхідно зробити для створення проекту і вихідного коду у вигляді наступної послідовності:

– провести первинний аналіз завдання, розробити формалізований опис задачі, обрати узагальнену модель рішення;

– написати функцію, яка відкриває текстовий файл в режимі читання, щоб забезпечити зчитування даних з текстового файлу до відповідних змінних. Необхідно правильно обрати типи даних для змінних, з урахуванням результатів, що отримані у ЛР 1;

– скористатися результатами лабораторної роботи 1 і виконати побудову таблиці (аналогічної ЛР 1) з використанням змінних, значення яких були зчитані з файлу, провести тестування результатів шляхом виведення таблиці на консоль, виконати налагодження програми;

– написати функцію, яка створює текстовий файл. Назву текстового файлу можна вести з консолі або ж ім’я файлу можна задати безпосередньо в

коді програми. Розроблена програма має записати до цього файлу таблицю, що отримана на попередньому етапі роботи. Необхідно переконатися що текстовий файл створений. Для цього необхідно відкрити його з використанням текстового редактору і переконатися у наявності таблиці у ньому;

– написати функцію, що створює бінарний файл, назву файлу можна ввести з консолі або ж задати безпосередньо в коді програми. Створена функція має записати до цього файлу кількість точок в таблиці, а також послідовність пар значень — значення аргументу та значення функції при заданому аргументу.

– перевірити результати роботи програми, пересвідчитися, що були створені бінарний та текстовий файли, переглянути вміст текстового файлу, оцінити розмір бінарного файлу і порівняти з текстовим файлом;

– оформити звіт до лабораторної роботи відповідно до вказаних вимог.

Теоретичні відомості ЛР 5

Під час запуску на виконання консольних програм, що передбачені у цьому курсі при виконанні лабораторних робіт, автоматично створюється три потоки: стандартний потік вводу stdin, стандартний потік виводу stdout і потік помилок stderr. Стандартний потік вводу автоматично підключається до клавіатури, потік виводу – до консолі (терміналу). Розробник ПЗ може написати програму, що утворює декілька додаткових потоків, переспрямувати стандартні потоки, наприклад для запису до файлу або певного апаратного пристрою.

Для роботи з потоками вводу-виводу у мові програмування С необхідно підлючити в програму файл заголовку *stdio.h*, що дає можливість використовувати стандартні бібліотеки для роботи з потоками. Він містить прототипи функції вводу-виводу, бібліотечні типи й макроси [8]. Отже потік – це джерело або одержувач даних, що є програмною сутністю. Бібліотека підтримує два види потоків, яким відповідають два типи даних: текстовий і двійковий (бінарний), що підтримуються у більшості ОС.

6.1 Фукнції для роботи із файлами

Для відкриття потоків до файлу використовується функція, що має прототип:

```
FILE* fopen ( const char* filename, const char* mode );
```

Дана функція відкриває файл із заданим ім'ям і повертає вказіваник на потік або ж повертає NULL якщо спроба відкриття виявилася невдалою.

Режими відкриття файлів наступні:

"r" – текстовий файл відкривається для читання;

"w" – текстовий файл створюється для запису;

"a" – текстовий файл відкривається або створюється для запису в кінець файлу;

"r+" – текстовий файл відкривається для виправлення (тобто для читання і для запису);

"w+" – текстовий файл створюється для виправлення; старий вміст (якщо він був) відсікається;

"a+" – текстовий файл відкривається або створюється для виправлення вже існуючої інформації і додавання нової в кінець файлу.

Якщо параметр режиму відкриття файлу доповнити буквою *b* (наприклад, *"rb"* або *"w+b"*), то це означатиме, що файл відкривається в бінарному режимі. Обмеження на довжину імені файлу задається константою FILENAME_MAX. Константа FOPEN_MAX обмежує число одночасно відкритих файлів.

Для роботи із файлами можуть використовуватися різні функції.

Функція

```
int remove ( const char* filename );
```

видаляє файл з вказаним ім'ям; подальша спроба відкрити файл з цим ім'ям викличе помилку. Функція повертає ненульове значення у разі невдалої спроби.

Функція

```
int rename ( const char*  oldname,  const char*  newname );
```

змінює ім’я файлу. Повертає ненульове значення у випадку, якщо спроба змінити ім’я виявилася невдалою. Перший параметр задає старе ім’я, другий параметр — нове ім’я.

Функція

```
FILE*  tmpfile ( void );
```

створює тимчасовий файл з режимом доступу «wb+», який автоматично видаляється при його закритті або звичайному завершенні програмою своєї роботи. Ця функція повертає потік або NULL якщо не змогла створити файл.

Функція

```
int  fprintf ( FILE*  stream, const char*  format, ... );
```

перетворює та записує дані в потік stream під управлінням *format*. Повертає значення – кількість записаних символів або, а у разі помилки повертає негативне значення.

Функція

```
int fscanf ( FILE* stream,  const char*  format, ... );
```

здійснює зчитування даних із вхідного файлу. Функція має невизначену кількість аргументів. Працює аналогічно функції *scanf()*.

Наведемо порядок роботи з функцією *fscanf()*. Для того щоб працювати із файлами необхідно в програмі оголосити вказівник на структуру FILE. Наприклад, у такий спосіб:

```
FILE * fp;
```

Надалі вказівник *fp* буде використовуватися для зчитування даних з файлу або запису даних у відповідний файл. Для вказівника можна обрати

будь-яке ім’я, що відповідає правилам складання ідентифікаторів. Для відкриття файлу необхідно використовувати функцію *fopen()*. Якщо файл, який відкривається в програмі, знаходиться в тому ж каталозі, що і програма, то необхідно вказати лише ім’я файлу, а також режим відкриття файлу. Наприклад, якщо такий файл має ім’я та розширення test_1.txt, тоді щоб відкрити цей файл, наприклад, в режимі читання необхідно виконати наступні дії:

```
fp = fopen ( "test_2.txt", "r");
```

Якщо файл, який необхідно відкрити, знаходиться в іншому каталозі ніж програма, тоді потрібно прописати повний шлях до цього файлу. При цьому, в якості елементів, що розділяють каталоги, необхідно використовувати два символи «зворотний слеш» для операційної системи Windows:

```
fp = fopen ( "D:\\projects\\files\\test_2.txt", "r" );
```

Якщо функція *fopen*() не змогла відкрити файл, то вона повертає значення NULL, яке присвоюється вказівнику fp. При зчитуванні даних із текстового файлу або для запису даних у текстовий файл можуть бути використані функції *fprintf()* та *fscanf()*, які схожі по своїй роботі із функціями *printf()* та *scanf()* за виключенням того, що потрібно вказувати в переліку параметрів вказівник на файл. Наприклад, якщо необхідно прочитати з текстового файлу значення змінної типу *double*, то це можна виконати наступним чином:

```
double x;
fscanf ( fp, ”%lf”, &x );
```

Для роботи з двійковими (бінарними) файлами використовуються функції *fwrite()* та *fread()*. Двійкові (бінарні) файли передбачають зберігання даних в тому вигляді, в якому вони зберігаються в пам’яті комп’ютера. При відкритті двійкового файлу необхідно до режиму відкриття файлу додавати символ *b*. Наприклад, для відкриття бінарного файлу *test_3.bin* в режимі «читання», при

умові що файл зберігається в тому ж каталозі, що і програма, яка його відкриває, необхідно записати:

```
fp = fopen ( "test_3.bin", "rb" );
```

Для закриття файлу необхідно використовувати функцію *fclose()*, наприклад:

```
fclose ( fp );
```

При роботі із текстовою інформацію, можуть бути використані відповідні функції, прототипи яких наводяться нижче.

Функція

```
int  fgetc ( FILE*  stream );
```

читає з файлу символ.

Функція

```
char*  fgets ( char*  s,  int n,  FILE*  stream );
```

читає з файлу рядок символів, але не більше за n символів.

Функція

```
int  fputc ( int  c,  FILE*  stream );
```

записує у файл символ.

Функція

```
int  fputs ( const char*  s,  FILE*  stream );
```

записує у файл рядок символів.

Функція

```
int  fseek ( FILE*  stream, long  offset,  int  origin );
```

встановлює позицію для *stream*. Подальше читання або запис проводитиметься з цієї позиції. У разі бінарного файлу позиція встановлюється із зсувом *offset* відносно початку, якщо параметр *origin* дорівнює *SEEK_SET*; або ж відносно

поточної позиції, якщо *origin* дорівнює *SEEK_CUR*; або відносно кінця файлу, якщо *origin* дорівнює *SEEK_END*.

Для текстового файлу параметр *offset* повинен бути нулем або значенням, отриманим за допомогою виклику відповідної функції.

При роботі з текстовим файлом параметр *origin* повинен бути рівним *SEEK_SET*.

Функція

```
long  ftell ( FILE*  stream );
```

повертає поточну позицію потоку *stream* або повертає значення «–1» у разі помилки.

Функція

```
int  fgetpos ( FILE*  stream,  fpos_t*  ptr );
```

записує поточну позицію потоку *stream* у вказівник *ptr* для подальшого використання її в *fsetpos()*. У разі помилки *fgetpos()* повертає ненульове значення;

Функція

```
int  fsetpos ( FILE*  stream,  const fpos_t*  ptr );
```

встановлює позицію в *stream*, читаючи її із вказівника *ptr*, де вона була записана раніше за допомогою *fgetpos()*. У разі помилки *fsetpos()* повертає ненульове значення.

6.2 Приклад використання функцій стандартної бібліотеки для роботи із файлами

Якщо необхідно прочитати текстовий файл і записати з нього значення, наприклад, до масиву, то потрібно відкрити файл в режимі читання. Для

вирішення цієї задачі зручно використовувати бібліотечну функцію *fscanf*, як це показано у прикладі коду:

```
if ((ptr_file_source_arg = fopen("source_file.txt", "r"))!=0) {
 while (fscanf(ptr_file_source_arg, "%d", &numbers_int_dest[i]) != EOF) {
        i++;
 }
 fclose(ptr_file_source_arg);
 } else {
    printf("Error! File cannot be opened 1 ");
    return 0;
 }
```

У даному прикладі коду відбувається читання файлу по рядках, що передбачені у файлі. Тобто у вихідному файлі *source_file.txt* нове значення має починатися з нового рядка. Читання відбувається до кінця файлу. Після читання одного рядка робиться збільшення змінної *i*. Запис відбувається до масиву. Файл має бути відформатований, кожний новий елемент має починатися з нового рядка. Приклад програми приведений далі.

```
#include <stdio.h>
#define ARRAY_SIZE 50
int main()
{
   int    numbers_int_dest[ARRAY_SIZE];
   double numbers_double_dest[ARRAY_SIZE];
   char char_array_dest[ARRAY_SIZE];
   char char_array_sourcef[ARRAY_SIZE];
   numbers_double_dest[49]='\0';
```

```
char_array_dest[49]='\0';
int i = 0;
FILE *ptr_file_source_arg;
FILE *ptr_file_txt_dest;
if ((ptr_file_source_arg = fopen("source_file.txt", "r"))!=0) {
   while (fscanf(ptr_file_source_arg, "%d", &numbers_int_dest[i]) != EOF)
   {i++;}
   fclose(ptr_file_source_arg);
  } else {printf("Error! File cannot be opened 1 ");
  return 0;
  }
 numbers_int_dest[i] = '\0';
 for (i = 0; numbers_int_dest[i] != '\0'; i++)
   printf("%d\n", numbers_int_dest[i]);
 for(i=0;numbers_int_dest[i]!='\0'; i++ )
   numbers_double_dest[i]=numbers_int_dest[i];
 for(i=0;numbers_double_dest[i]!='\0'; i++ )
   printf("%lf\n", numbers_double_dest[i]);
 if ((ptr_file_source_arg = fopen("testfile.txt", "r"))!=0) {
   fscanf(ptr_file_source_arg, "%[^\n]", char_array_dest);
   fclose(ptr_file_source_arg);
  } else {printf("Error! File cannot be opened 2 ");
  return 0;
  }
 for (i = 0;  char_array_dest[i] != '\0'; i++)
   printf("%c\n",  char_array_dest[i]);
```

```
    if ((ptr_file_txt_dest = fopen("dest_file.txt", "w"))!=0) {
        printf("Enter a sentence:\n");
        fgets(char_array_sourcef, sizeof(char_array_sourcef), stdin);// to array
        fprintf(ptr_file_txt_dest, "%s", char_array_sourcef);
        fclose(ptr_file_txt_dest);
     } else {printf("Error! File cannot be opened 3 ");
     return 0;
    }
    return 0;
}
```

Текстові файли зберігають всі дані як текст, у тому числі і такі, що описані як числа або цифри. Це означає, що якщо, наприклад, ми записуємо ціле число 12345678 у файл, то до файлу записується вісім символів, де кожен символ потребує для свого зберігання 1 байт, тобто в даному прикладі буде записано 8 байт даних в поточний тектовий файл. На це не впливає тип даних змінної де зберігалося це число. Наприклад, у програмі може бути реалізовано змінна цілого типу, а у текстовий файл вона буде записана як текст. Виведення та введення даних може бути форматованим, тобто, щоразу коли ми зчитуємо число з файлу або записуємо у файл відбувається трансформація числа відповідно до формату, що вказано у функції *fscanf()*.

6.3. Контрольні запитання до лабораторної роботи № 5

1. Наведіть шаблон, призначення та опис роботи функції *fopen()*.
2. Наведіть шаблон, призначення та опис роботи функції *fscanf()*.
3. Наведіть шаблон, призначення та опис роботи функції *fprintf()*.
4. Наведіть шаблон, призначення та опис роботи функції *fgets()*.
5. Поясніть особливості текстового файлу і бінарного файлу.

7. ЛАБОРАТОРНА РОБОТА № 6 «ТЕХНОЛОГІЇ РОЗРОБКИ З ВИКОРИСТАННЯМ Make КОМПІЛЯТОРА GCC ДЛЯ POSIX СУМІСНИХ ОПЕРАЦІЙНИХ СИСТЕМ»

Мета лабораторної роботи полягає у набутті знань, умінь та навичок з технології розроблення системного програмного забезпечення з використанням мови С для POSIX (Portable Operating System Interface for uniX) сумісних (сертифікованих) операційних систем. Під час використання С як мови програмування застосовуються знання основ архітектури мікропроцесора, загальної будови операційної системи та прийнятного стилю вихідного коду. Також, лабораторна робота дає основні навички користування ОС Linux.

Робота виконується з використанням бібліотек мови С для POSIX ОС та автоматизації розробки з використанням Makefile. Для виконання курсу ЛР потрібен комп'ютер, на якому встановлена POSIX операційна система, або віртуальна машина з POSIX OS, або система контейнеризації. Особливих вимог до потужності персонального комп'ютера (ПК) не визначається.

Для підготовки ЛР № 6–8 може бути використаний будь-який сучасний ПК на основі мікропроцесора AMD64 (Intel® 64) або ARM. Наданий вихідний код протестований для операційної системи Linux.

Вхідні дані ЛР 6

Для виконання лабораторної роботи потрібно скористатися результатами лабораторної роботи № 5, а саме: текстовим файлом і вихідним кодом. У вихідному текстовому файлі знаходиться інформація, що призначена для виконання ЛР 5.

Вихідні дані ЛР 6

Каталог з вихідним кодом роботи, Makefile. Текстовий файли з результатом виконаним відповідно до завдання та бінарний файл з результатом.

Завдання

1. Створити каталог для проекту, розташувати його у файловій структурі POSIX операційної системи, або віртуальній машині з POSIX OS, або у системі контейнеризації. Вибір здійснюється користувачем.

2. Додати до каталогу файл з вихідним кодом ЛР 5 (*main.c*), створити окремо файл вихідного коду бібліотеки і файлу заголовку для функцій проекту. Надати їм назви, наприклад *lib.c*, *lib.h.*

3. Відокремити функції, "вирізати" їх з файлу *main.c* і додати реалізацію функцій до файлу *lib.c*, прототипи функцій до *lib.h.*

4. Створити Makefile текстовим редактором, зібрати проект, запустити його і переконатися у правильності роботи програми.

5. Перевірити результати роботи програми. Пересвідчитися, що були створені бінарний та текстовий файли. Переглянути вміст текстового файлу. Оцінити розмір бінарного файлу.

6. Зчитати з текстового файлу назву групи та номер бригади і вивести їх на консоль, до текстового файлу і до бінарного файлу.

7. Оформити звіт до лабораторної роботи відповідно до вказаних вимог. Звіт і вихідний код розташувати на https://github.com/.

Програма проведення експерименту ЛР 6

1. Залогінитися до POSIX операційної системи, або віртуальної машини з POSIX OS, або у систему контейнеризації через термінал. Створити каталог, користуючись командним рядком або графічним інтерфейсом для проекту, розташувати його у файловій структурі.

2. Додати до каталогу файл з вихідним кодом ЛР 5 (*main.c*). Користуючись текстовим редактором операційної системи створити окремо файл вихідного коду бібліотеки і файлу заголовку для функцій проекту. Надати їм назви, наприклад *lib.c, lib.h.* Оптимальним для початківця є текстовий редактор *Nano*.

3. Відокремити функції, “вирізати” їх з файлу *main.c* і додати реалізацію функцій до файлу *lib.c*, прототипи функцій до *lib.h.*

4. Створити Makefile текстовим редактором, зібрати проект, запустити його і переконатися у правильності роботи програми.

5. Перевірити результати роботи програми. Пересвідчитися, що були створені бінарний та текстовий файли. Переглянути вміст текстового файлу. Оцінити розмір бінарного файлу.

6. У випадку виникнення помилок на етапі компіляції або виконання, виявити їх, внести виправлення до відповідних файлів, провести збірку ще раз.

7. Оформити звіт до лабораторної роботи відповідно до вказаних вимог. Звіт і вихідний код розташувати на https://github.com/.

Теоретичні відомості ЛР 6

Системне програмування є широкою галуззю програмування, один з напрямків якого передбачає розробку операційних систем. Також до цього напрямку програмування відноситься розробка модулів ядра, драйверів. На теперішній час для вирішення практичних завдань із розробки вбудованих систем, серверів тощо, доцільно використовувати POSIX (Portable Operating System Interface for uniX) операційні системи.

Операційні системи POSIX є багатозадачними системами з розділенням часу. Це можуть бути різні ОС, що сертифіковані за стандартами POSIX, або суміжними з цим стандартом, що мають однакову оболонку, інтерфейси ядра, схему інтерфейсів користувача, програмний інтерфейс (інтерфейс системних викликів), а також схожі принципи побудови архітектури ОС у цілому.

До операційних систем POSIX можна віднести: UNIX та її варіанти, Linux, Solaris тощо. На теперішній час розповсюдження Linux значною мірою зумовлено відкритим програмним кодом. Ця ОС успішно застосовується на персональних комп’ютерах, великих серверах, потужних обчислювальних

кластерах, у мережевому обладнанні (маршрутизаторах), смартфонах, і навіть у вбудованих пристроях [1].

Як звичайні ОС, операційні системи POSIX мають у своєму складі ядро ОС, що виконує певний перелік завдань. Крім того, ці ОС мають командну оболонку, роль якої можуть виконувати традиційні командні інтерпретатори – sh, ksh, csh, tcsh для bash тощо [1].

Галузь системного програмування охоплює розробку ПО на рівні ядра і оболонки. На першому рівні використовуються мова програмування С(С++), у другому варіанті використовуються типова мова оболонки або мова програмування Python. В деяких операційних системах по замовчуванню не встановлені програми для збірки і компіляції. Для їх інсталяції використовують команди оболонки, наприклад, для ОС Debian і інших систем, що походять від неї, використовуються наступні команди:

```
sudo apt update

sudo apt-get install gcc.

sudo apt install make

sudo apt-get install build-essential
```

Будуть інстальовані наступні елементи: cc-5-locales, g++-multilib, g++-5-multilib, gcc-5-doc, gcc-multilib, autoconf, automake, libtool, flex, bison, gdb, gcc-doc, gcc-5-multilib.

Послідовність розробки програми з використанням компілятору GCC у найбільш простому випадку включає кроки, що показані далі.

На першому кроці, з використанням текстового редактору створюємо файл hello.c, що містить вихідний код програми, наприклад [1]:

```
/ * Hello.c * /

#include <stdio.h>
```

```
int main (void){

printf ( "Hello World \ n");}
```

Щоб відкомпілювати hello.c достатньо набрати у командному рядку наступне:

```
$ gcc   -o    hello    hello.c
```

Якщо вихідний код написаний без синтаксичних помилок, то компілятор завершить свою роботу без будь-яких повідомлень, в результаті утворюється файл - hello. Цей файл містить машинний код програми (executable files) або є інша назва - бінарниий файл (binary files).

Опція -о компілятора GCC вказує на те, яким має бути ім'я вихідного файлу. Якщо не вказати опцію -о, то бінарнику буде присвоєно ім'я a.out. Щоб запустити програму набираємо в командному рядку наступну команду:

$ ./hello,

результатом буде Hello World на терміналі.

Для мультіфайлового програмування потрібно зробити лінкування декількох файлів. Отже, щоб побудувати багатофайлову програму, треба спочатку отримати об'єктний код з кожного вихідного файлу окремо. Кожен такий код буде представляти собою об'єктний модуль. Кожен об'єктний модуль записується до окремого об'єктного файлу. Потім об'єктні модулі треба скомпонувати в один бінарник з точкою входу.

У Linux для лінкування використовується програма ld. Програма GCC самостійно викликає його з потрібними опціями. Для прикладу створимо перший файл з ім'ям main.c:

```
/ * main.c * /

  int main() {
     int steps=3;
```

```
    func(steps);
    return 0;
}
```

Тепер створимо ще один файл, що містить бібліотечну функцію. Назва файлу lib.c:

```
#include <stdio.h>
void func(int steps){
    int i = 0;
  for (; i<steps;i++){
            printf("hello\n");
  }
```

Функція main () викликає функцію *func*(), що знаходиться в іншому файлі. Опція -c компілятора GCC змушує його відмовитися від лінковки після компіляції. Якщо не вказувати опцію -o, то в імені об'єктного файлу розширення *.c* буде замінено на *.o* (звичайні об'єктні файли мають розширення *.o*):

```
$ gcc -c main.c

$ gcc -c  lib.c

$ ls

lib.c lib.o main.c main.o
```

Отже, ми отримали два об'єктних файла. Далі їх треба об'єднати до одного:

```
$ gcc -o hello main.o lib.o

$ ls

hello * lib.c lib.o main.c main.o
```

$ ./hello, результатом буде

Hello World.

Компілятор "побачив", що замість вихідних файлів (з розширенням *.c*) йому надаються об'єктні файли (з розширенням *.o*) і відреагував згідно ситуації: викликав лінкувальник з потрібними опціями.

Особливість об'єктних файлів полягає у тому, що вони зберігають в таблиці символів імена деяких позицій (глобально оголошених функцій, наприклад). В процесі лінковки відбувається стикування імен та перерахунок позицій, що дозволяє декільком об'єктним файлам об'єднатися до одного бінарника. Якщо викликати nm для файлу hello.o, то побачимо наступну картину:

```
$ Nm hello.o

U printf

00000000 T print_hello
```

Важливим є те, що в об'єктному файлі збереглася інформація про імена. Своя інформація є і в файлі main.o:

```
$ Nm main.o

00000000 T main

    U print_hello
```

Звичайно, у професійній розробці програмного забезпечення використовується автоматична збірка. Розглянемо сутність автоматичної збірки. У попередньому розділі для створення бінарників з двох вихідних файлів потрібно набрати три команди. Утиліта make, що працює за своїми власними сценаріями дає можливість автоматизувати процес виконання команд. Сценарій записується в файлі з ім'ям Makefile і поміщається в репозиторій (робочий каталог) проекту. Формат Makefile використовується не тільки на Unix-системах, а в інших ОС. Процес виконання make називають

складанням проекту, а сама утиліта make відноситься до розряду програм — збирачів проектів.

Файл Makefile складається з трьох елементів: коментарі, макровизначення і цільові зв'язки (або просто зв'язки). У свою чергу зв'язки складаються теж з трьох елементів: мети, залежності та правила. Сценарій make використовує однорядкові коментарі, що починаються з літери # (решітка). Макроси Makefile схожі з макроконстантами мови С. Простий приклад Makefile у загальному вигляді наданий далі:

```
мета: залежності
    [tab] команда
```

Разом це можна описати так:

- що потрібно зробити (мету);

- що для цього потрібно (залежності);

- як це зробити (правила).

У якості мети виступає ім'я або макроконстанта. Залежності – це список файлів і цілей, розділених між собою. Правила – це команди, що передаються оболонці.

Розглянемо приклад сценарію збірки для розглянутого в попередньому розділі мультифайлового проекту. Для цього створіть файл з ім'ям Makefile, зміст якого наданий далі:

```
binary.exe:main.o lib.o
    gcc -o binary.exe main.o lib.o
main.o:main.c
    gcc -c main.c
lib.o:lib.c
    gcc -c  lib.c
clean:      rm -f *.o binary.exe
```

Для мови програмування C++ аналогічний файл має наступний синтаксис:

```
all: hello

hello: main.o lib1.o lib2.o

    g++ main.o lib1.o lib2.o -o hello

main.o: main.cpp

    g++ -c main.cpp

lib1.o: lib1.o.cpp

    g++ -c lib1.o.cpp

lib2.o: lib2.cpp

    g++ -c lib2.cpp

clean: rm -rf *.o hello
```

Зверніть увагу, що в кожному рядку перед викликом gcc(g++), а також в рядку перед викликом rm стоять табуляції. Формат Makefile вимагає, щоб кожне правило починалося з табуляції. Makefile може починатися як з великої так і з малої літери. Прийнято починати з великої букви, щоб він не переміщувався з іншими файлами проекту, а стояв "у списку перших". У даному типі файлу може бути використані макровизначення, наприклад:

```
# Це коментар, де використано макровизначення

CC=g++

#У CFLAGS знаходяться прапори

CFLAGS=-c -Wall

all: hello

hello: main.o lib1.o lib2.o
```

```
	$(CC) main.o lib1.o lib2.o -o hello

main.o: main.cpp

	$(CC) $(CFLAGS) main.cpp

lib1.o: lib1.cpp

	$(CC) $(CFLAGS) lib1.cpp

	lib2.o: lib2.cpp

	  $(CC) $(CFLAGS) lib2.cpp

clean: rm -rf *.o hello
```

Перший рядок – коментар, що починається з символу # (решітка) і закінчується символом нового рядка. Далі по порядку йдуть чотири зв'язки:

- зв'язка для компонування об'єктних файлів main.o і hello.o;
- зв'язка для компіляції main.c;
- зв'язка для компіляції hello.c;
- зв'язка для очищення проекту.

Очищення проєкту буває потрібним в декількох випадках:

- для очищення готового проекту від всього зайвого;
- для збирання проекту (коли в проект додаються нові файли або коли змінюється сам Makefile;
- в будь-яких інших випадках, коли потрібна повна зібрання (наприклад, для вимірювання часу повного складання).

Формат запуску утиліти make наступна:

```
make [опції] [мети …].
```

Якщо викликати make без визначення мети, то буде виконана зв'язка (з усіма залежностями) і збірка завершиться:

```
$ make
```

```
gcc -c main.c

gcc -c hello.c

gcc -o hello main.o hello.o

$ ls

hello * hello.c hello.o main.c main.o Makefile

$ ./hello

Hello World

$
```

Для очищення проекту потрібно викликати команду:

```
$ Make clean

rm -f * .o hello

$ ls

hello.c main.c Makefile

$
```

В даному випадку ми вказали мету безпосередньо в командному рядку. Цільова зв'язка *clean* містить порожній список залежностей, виконується тільки одне правило. Потрібно "чистити" проєкт, коли змінюється список вихідних файлів або коли змінюється сам Makefile.

7.1. Контрольні запитання до лабораторної роботи № 6

1. Наведіть опис каталогів POSIX сумісних (сертифікованих) ОС.
2. Назвіть основні команди оболонки для управління і роботи у POSIX сумісних (сертифікованих) ОС.

3. Права доступу до файлів у типових POSIX сумісних (сертифікованих) ОС. Способи відеозображення і описання.

4. Основні команди для всіх етапів розробки програм у GCC компіляторі.

5. Основні правила розробки Makefile.

8. ЛАБОРАТОРНА РОБОТА № 7 «СТВОРЕННЯ СТАТИЧНИХ І ДИНАМІЧНИХ БІБЛІОТЕК З ВИКОРИСТАННЯМ Makefile КОМПІЛЯТОРА GCC»

Мета роботи: полягає у оволодінні практичними навичками розробки і використання статичних і динамічних бібліотек на системному рівні з використанням мови С для POSIX OS, на прикладі ОС Linux та автоматизації розробки з використанням Makefile.

Перед початком розробки програм потрібно вивчити: основи синтаксису статичних і динамічних бібліотек; організацію і структуру файлової системи POSIX OS, обмеження на імена файлів; типи файлів, каталоги і посилання, синтаксис Makefile.

Вхідні дані ЛР 7

Для виконання лабораторної роботи потрібно скористатися результатами лабораторної роботи № 6, а саме: текстовим файлом, вихідним кодом, Makefile. У вихідному текстовому файлі знаходиться інформація, що призначена для виконання ЛР 7.

Вихідні дані ЛР 7

Два репозиторія, один з яких містить проект на основі статичних бібліотек, інший містить проект на основі динамічних бібліотек. У кожному репозиторії має бути текстовий файли з результатом, виконаним відповідно до завдання, бінарний файл з результатом.

Завдання

1. Створити два каталоги для проектів для статичних бібліотек і динамічних бібліотек, розташувати їх у файловій структурі POSIX операційній

системи, або віртуальній машині з POSIX OS, або у системі контейнеризації. Вибір здійснюється користувачем.

2. Додати до каталогів файли результатів ЛР 6, а саме: *main.c, lib.c, lib.h,* Makefile.

3. Внести зміни до Makefile, у кожному з двох репозитаріїв у такий спосіб, щоб були створені проекти на основі статичної і динамічної бібліотеки, внести відповідні змінні до файлів вихідного коду.

4. Оформити звіт до лабораторної роботи відповідно до встановлених вимог. Звіт і вихідний код розташувати на https://github.com/.

Програма проведення експерименту ЛР 7

1. Залогінитися до POSIX операційної системи, або віртуальної машини з POSIX OS, або у систему контейнеризації через термінал. Створити два каталогу для проектів статичних бібліотек і динамічних бібліотек.

2. Додати до каталогів файли з вихідним кодом ЛР 6. На даному етапі перевірити коректність первинних результатів, провести збірку проектів з використанням Makefile.

3. Користуючись текстовим редактором операційної системи внести зміни до Makefile і вихідного коду, з метою отримання двох проектів з статичними і динамічними бібліотеками. Оптимальним для початківця є текстовий редактор Nano.

4. Перевірити результати роботи двох проектів. Пересвідчитися, що були створені бінарний та текстовий файли.

5. Переглянути вміст текстового і бінарного файлу. Оцінити розмір файлів.

6. У випадку виникнення помилок на етапі компіляції або виконання виявити їх, внести виправлення до відповідних файлів, провести збірку ще раз.

7. Оформити звіт до лабораторної роботи відповідно до встановлених вимог. Звіт і вихідний код розташувати на https://github.com/.

Теоретичні відомості ЛР 7

У програмуванні широко застосовуються статичні і динамічні бібліотеки, наприклад, у операційних системах родини Windows або Linux. Під час збірки програми, що містить статичну бібліотеку, бінарний код статичної бібліотеки стає частиною програмного файлу, що виконується. У Windows статичні бібліотеки мають розширення *.lib*. У операційній системі Linux статичні бібліотеки мають розширення *.a*. Однією з переваг статичних бібліотек у проекті є те, що файл програми, що виконується, вже містить всі потрібні складові бібліотеки і не потрібно додаткових файлів. З іншого боку, це також є недоліком рішення, оскільки розмір файлу, що виконується, може необґрунтовано збільшуватися. Крім того, виникають труднощі під час підтримки програми.

Динамічна бібліотека може містити такі самі функції як і статична, при цьому ці дві технології мають суттєву відмінність. Звичайно динамічна бібліотека формується як окремий файл, що завантажується під час виконання програми тільки у тому випадку, коли у ході виконання використовується функція (метод, клас, структура, тощо) цієї бібліотеки.

У Windows динамічні бібліотеки мають розширення *.dll* «dynamic link library» і можуть знаходитися у спеціальних каталогах операційної системи або у тому каталозі, де знаходиться файл програми. Особливістю динамічних бібліотек Windows є те, що вони є також складовою частиною операційної системи.

Отже, при збірці програми, що використовує динамічну бібліотеку, ця бібліотека не стає частиною файлу програми і залишається окремим модулем.

У Linux динамічні бібліотеки мають розширення *.so* «shared object». Однією з переваг динамічних бібліотек є можливість більш простого внесення змін до програми. Крім того, різні програми можуть спільно використовувати одну копію динамічної бібліотеки, саме це характерно для системного рівня

Windows. Це значно економить дискову пам'ять і спрощує процес оновлення і супроводження.

Відмінністю програм з динамічними бібліотеками є особлива структура проекту і особливості у вихідному коді. Це обґрунтовано потребою явно підключати і взаємодіяти з динамічною бібліотекою. Розглянемо технологію і приклад створення програми, що використовує статичну бібліотеку для Linux. Будемо виконувати це на прикладах, що наведені у [10].

Створимо свою власну бібліотеку, що має функції: *h_world*() і *g_world*(), і виводять на екран "Hello World" і "Goodbye World" відповідно. Почнемо зі статичної бібліотеки.

Створимо файл world.h [10]:

```
/ * World.h * /

void h_world (void);

void g_world (void);
```

Створимо файл h_world.c:

```
/ * H_world.c * /

#include <stdio.h>

#include "world.h"

void h_world (void)

{

    printf ( "Hello World \ n");

}
```

Далі потрібно створити файл g_world.c, що містить реалізацію функції *g_world*() [9]:

```
    / * G_world.c * /
```

```
#include <stdio.h>

#include "world.h"

void g_world (void){

 printf ( "Goodbye World \ n");

}
```

Для наочності і демонстрації статичних бібліотек, цей вихідний код розділено на два файли.

Файл *main.c*, що використовує динамічні бібліотеки, містить наступний код [10]:

```
/ * Main.c * /

#include "world.h"

int main (void){

        h_world ();

        g_world ();

}
```

Щоб компілятор “знав” де шукати бібліотеки у Makefile треба визначити каталог, у якому містяться бібліотеки і список цих бібліотек. Каталог з бібліотеками вказується ключем *-L*. У нашому випадку бібліотека знаходиться в поточному каталозі, значить шлях до неї буде у вигляді точки (*-L*). Використані бібліотеки перераховуються через ключ *-l*, після якого вказується назва бібліотеки без префікса *lib*.

Далі напишемо Makefile за зразком, вказаним у [10]:

```
# Makefile for World project

binary: main.o libworld.a

        gcc -o binary main.o -L. -lworld
```

```
main.o: main.c
	gcc -c main.c
libworld.a: h_world.o g_world.o
	ar cr libworld.a h_world.o g_world.o
h_world.o: h_world.c
	gcc -c h_world.c
g_world.o: g_world.c
	gcc -c g_world.c
clean:
	rm -f * .o * .a binary
```

У прикладі нижче для наочності у репозиторії присутні дві статичні бібліотеки.

```
binary:main.o test_lib.a
  gcc -o binary main.o -L. -l:   test_lib.a
main.o:main.c
  gcc -c main.c
test_lib.a:   f1.o   f2.o
  ar cr test_lib.a f1.o f2.o
f1.o:   f2.c
  gcc -c f1.c
f2.o:   f2.c
  gcc -c f2.c
```

```
clean:
	rm -f *.o *.a binary
```

Покажемо команди, що треба виконати для збірки проекту з динамічними бібліотеками і результати роботи [10]:

```
$ make
gcc -c main.c
gcc -c h_world.c
gcc -c g_world.c
ar cr libworld.a h_world.o g_world.o
gcc -o binary main.o -L. -lworld
```

Для перевірки результату треба запустити програму, яка виконує наступні команди у терміналі:

```
$ ./binary
Hello World
Goodbye World
```

В наведеному прикладі, після команди *make* з'явилися три нових повідомлення, а саме: опції *-l* і *-L* компілятора, а також команда *ar*.

Для того, щоб створити і використовувати динамічну (спільно використовувану) бібліотеку, потрібно переробити у попередньому проекті Makefile [10].

```
# Makefile for World project
binary: main.o libworld.so
gcc -o binary main.o -L. -lworld -Wl, -rpath ,.
main.o: main.c
```

```
gcc -c main.c

libworld.so: h_world.o g_world.o

gcc -shared -o libworld.so h_world.o g_world.o

h_world.o: h_world.c

gcc -c -fPIC h_world.c

g_world.o: g_world.c

gcc -c -fPIC g_world.c

clean:

rm -f * .o  *.so binary
```

Правило для збірки *binary* тепер містить опцію *-Wl, -rpath*. Вона дає можливість передати лінкувальнику опцію *-rpath* з аргументами.

У нашому випадку ми передаємо лінковщик опцію *-rpath* з аргументом, де точка позначає поточний каталог. Що означає опція *-rpath*? Лінкувальник шукає бібліотеки в місцях, що передбачені у операційної системі по замовчуванню. Звичайно це каталоги */ lib* та */ usr / lib*, іноді */ usr / local / lib*. Опція *-rpath* просто додає до цього списку ще один каталог. У нашому випадку це поточний каталог. Без вказівки опції *-rpath* програма не запуститься через відсутність динамічної бібліотеки.

Отже, статична бібліотека – це колекція об'єктних файлів, які приєднуються до програми під час компонування програми.

У динамічній бібліотеці об'єктні файли підключаються тільки у тому випадку, коли здійснюється виклик функцій, що належать до динамічної бібліотеки. Приєднання цих об'єктних файлів здійснює системний динамічний завантажувач під час запуску програми.

8.1. Контрольні запитання до лабораторної роботи № 7

1. Базовий перелік команд оболонки для управління і роботи у POSIX сумісних (сертифікованих) ОС.
2. Права доступу до файлів у типових POSIX сумісних (сертифікованих) ОС. Способи відображення і описання. Способи зміни прав доступу.
3. Основні правила розробки Makefile для створення статичних і динамічних бібліотек.
4. Відмінності статичних і динамічних бібліотек у POSIX сумісних (сертифікованих) ОС.
5. Основні команди для всіх етапів розробки програм у GCC компіляторі.
6. Описання роботи розробленої програми на алгоритмічному рівні.
7. Синтаксис класу у C++, створення екземпляру класу, рівні доступу, управління рівнями доступу, доступ до методів екземпляру класу.

9. ЛАБОРАТОРНА РОБОТА № 8 «МЕТОДИ І ТЕХНОЛОГІЇ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ»

Мета роботи: отримати навички роботи зі складними програмними системами, що містять елементи паралельного програмування. Лабораторна робота виконується при успішному захисті ЛР № 1–5 і по узгодженню з викладачем.

Вхідні дані ЛР 8

Технічне завдання на програмну систему з елементами паралельного програмування, що надана далі у 4 варіантах. Склад системи, алгоритм роботи програмної системи, список команд, вимоги системи візуалізації (текстовий інтерфейс користувача або графічний інтерфейс, по узгодженню з викладачем), вимоги до лог файлу.

Вихідні дані ЛР 8

Прототип програмної системи у вигляді вихідного коду, скріншоти роботи системи.

Завдання. Варіант 1. Опис технологічного процесу

Система обслуговування клієнтів у сфері харчування

Дана система складатиметься з комерційного приміщення, поділеного на три зони: зона обслуговування, зона приготування замовлень та зона видачі замовлень.

Зона обслуговування складатиметься з двох терміналів, клієнти закладу можуть паралельно виконувати замовлення за їх допомоги. Зона приготування замовлень складається з екрану замовлень, відповідно до якого кухарі виконують замовлення на трьох станціях приготування паралельно.

Зона видачі складатиметься зі столу збору замовлень, здатного послідовно розподілити замовлення між точками видачі. Останні мають змогу обслуговувати клієнтів паралельно.

Система обслуговування має наступні правила:

– каси самообслуговування та точки видачі здатні виконувати лише дві задачі паралельно;

– клієнти мають право робити лише один вид замовлення — комбо-меню “Класичне”, яке складається з бургеру, порції картоплі-фрі та напою “Нюка Кола”;

– станції приготування замовлення готують його компоненти окремо, незалежно від замовлення;

– стіл збору замовлень синхронізує процес приготування різнопланової їжі.

Алгоритм системи. Варіант 1

Розроблювана система представляє собою замкнений цикл та працює відповідно до наступного алгоритму:

1. Покупець потрапляє у комерційну зону та підходить до вільної каси самообслуговування, якщо вільної каси не має, то він очікує в черзі.

2. Потрапивши до каси самообслуговування покупець робить замовлення на комбо - меню “Класичне” та переходить в чергу очікування на замовлення до однієї з точки видачі.

3. Замовлення потрапляє до екрану замовлень та розподіляється по чергам трьох станцій приготування: бургерів, картоплі та напоїв.

4. По мірі того, як окремі компоненти виконуються, вони збираються у повноцінне замовлення.

5. Як тільки замовлення стає повністю зібраним, воно відправляється на відповідну точку видачі.

6. Покупець забирає замовлення з точки видачі та виходить з комерційної зони.

Список команд. Варіант 1

1. Вхід покупців.
2. Очікування покупця n в черзі.
3. Процес створення замовлення покупцем n.
4. Потрапляння замовлення в список замовлень.
5. Очікування замовлення на бургер в черзі на приготування.
6. Приготування бургера.
7. Очікування замовлення на картоплю в черзі на приготування.
8. Приготування картоплі.
9. Очікування замовлення на напій в черзі на приготування.
10. Наливання напою.
11. Очікування в черзі на збирання.
12. Збирання замовлення.
13. Видача замовлення покупцю n.
14. Вихід покупців.

Вимоги до візуалізації. Варіант 1

Програма являє собою настільний або веб-додаток, у якому представлена вся система обслуговування клієнтів. Елементами даної системи є:

- черги до кас;
- каси самообслуговування, де покупці роблять замовлення;
- панель зі списком всіх замовлень;
- станції приготування та черги до них: плита для бургерів, фритюр для картоплі, видача напоїв;

- точка збирання (пакування) замовлень та її черга;
- точки видачі замовлень;
- черги до точок видачі.

Додатковим елементом є кнопка, яка додає нового покупця і він автоматично заходить до кафе. Всі інші дії також виконуються автоматично.

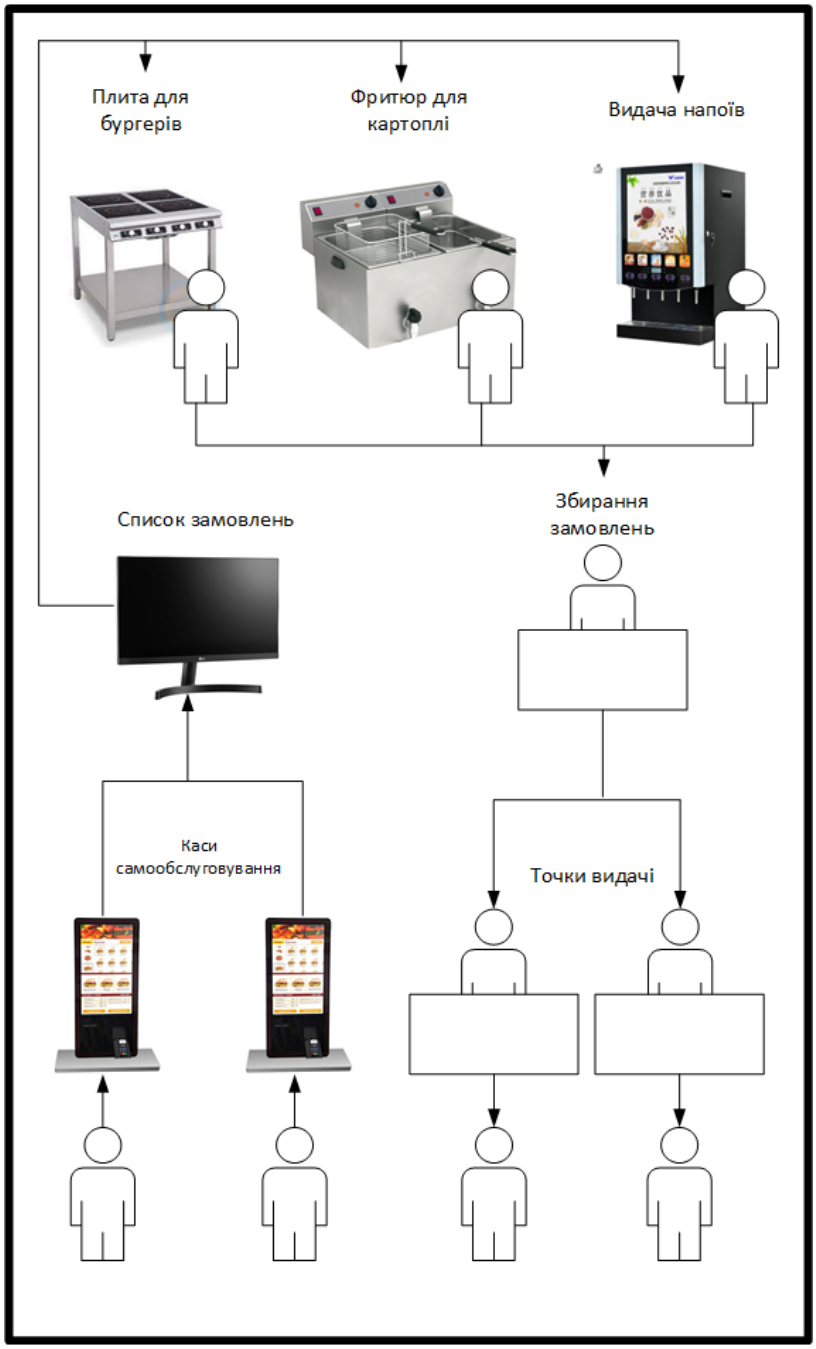


Рис. 21 — Візуальне представлення системи

Вимоги до лог файлу. Варіант 1

Вимоги складені у наступному варіанті:

– лог файл зберігається в оперативній пам’яті;

– максимальний розмір лог файлу складатиме 64 МБ;

– лог файл має складатися с UTF-8 символів;

– кожна строка логу має відображати конкретну подію в системі обслуговування;

– кожне повідомлення з логу має супроводжуватись назвою події, параметрів виконання, та часової мітки в часовій зоні UTC;

– система має містити інтерфейс для експорту логу у файл;

– назва файлу має бути diner-service.log.

Завдання. Варіант 2. Опис технологічного процесу

Система управління тиром

У даній системі буде територія, в якій буде розташовано певна кількість цілей для кожного із п’ятьох осіб, які будуть стріляти. В цій системі буде розташовано 45 цілей для всіх учасників в цілому, для кожного буде 9 цілей, відстань від стрілка до цілі десять метрів. Існує певні правила у роботі тиру:

• кожен учасник має десять пострілів;

• не стріляти, доки цілі виставляються;

• після десяти пострілів, учасник покидає територію тиру (з призом);

• при зайнятості усіх місць, учасник стоїть у черзі;

• час на заряджання зброї після стрілка 30 секунд;

• час заміни цілей кожного стрілка 30 секунд;

• після того як стрілок прийшов очікування інших 30 секунд

Розподіл між учасниками в тирі повинен бути описаний на основі функції від навантаження і часу очікування. Повне проходження циклу буде займати такий час:

Загальний час проходження групи стрілків = (2 хвилини + 30 секунд * *кількість зброї в черзі на перезарядку* + 30*(кількість учасників–1)+30 секунд).

Алгоритм системи. Варіант 2

Алгоритм управління матиме такі правила:

1. Перед сесією стрільбі учасники очкують заповнення усі х місць(або якщо нових не має починають сесію)

2. Стрілець має зробити десять пострілів.

3. Коли стрілки закінчили стріляти то цілі вони оновлюються.

4. Після збиття п’яти цілей, видається приз.

5. Після однієї групи, починає збиратися наступна.

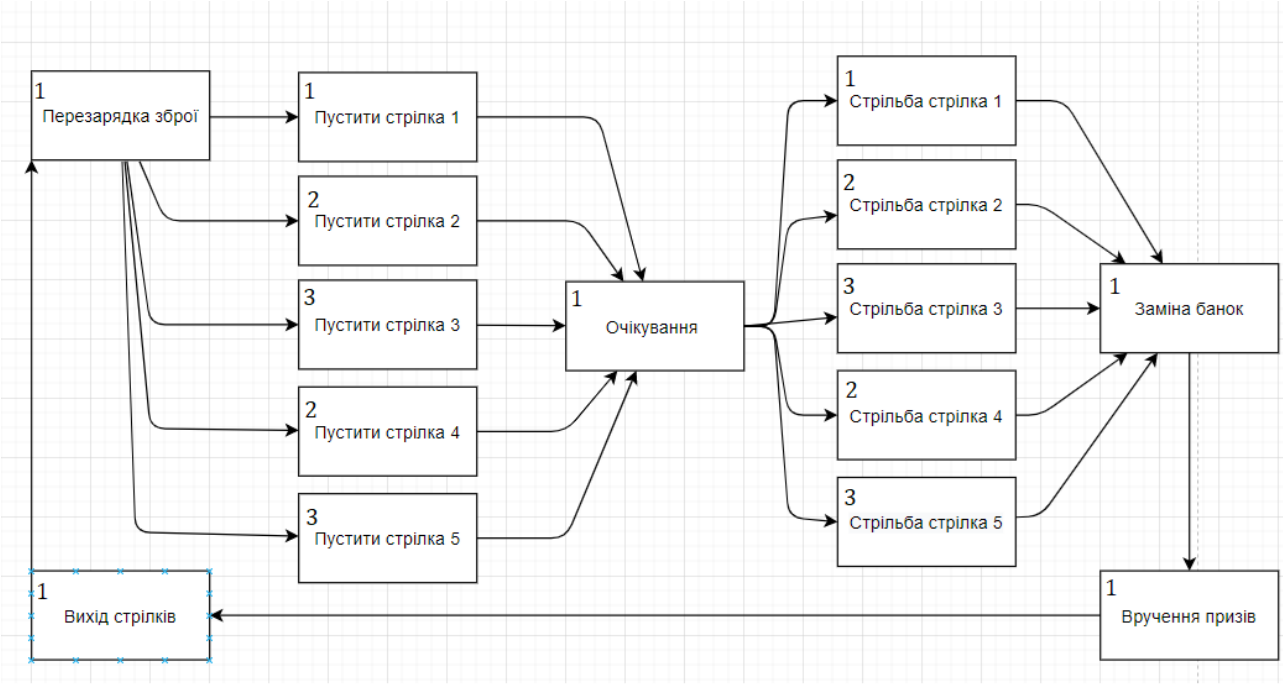


Рис. 22 — Паралельно-послідовний алгоритм системи управління тиром

Список команд. Варіант 2

1. Пустити стрілка
2. Очікування
3. Стрільба
4. Заміна банок

5. Зупинити стрільбу

6. Вручення призу

7. Перезарядка зброї

8. Вихід стрілка

Вимоги до візуалізації. Варіант 2

Програма являє собою одне вікно, у якому буде представлена вся система керування. Елементами даної системи є:

• 5 місць для пострілів, де можуть стріляти учасники (квадрати на схемі без символів);

• кожен з учасників має 9 мішеней (круглі на схемі)

• після пострілів проводиться перезарядка зброї інструктором (квадрат на схемі з символом)

• панель з кнопкою, яка викликає переривання.

Кнопка зупинки пострілів, яка знаходиться в нижній половині екрана (справа внизу). Стрілки можуть знаходитися тільки за стійкою. При натискання на кнопки, зупиняються постріли. При заміні банок, постріли зупиняються.

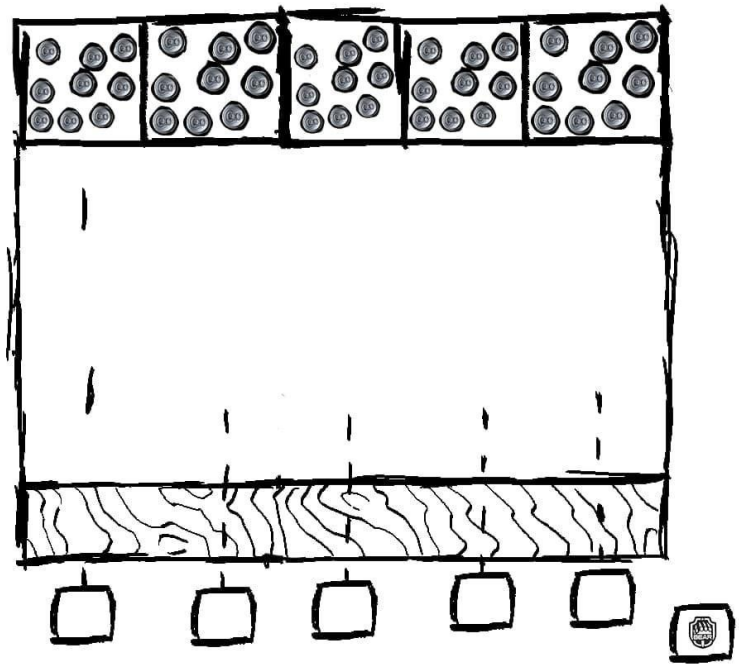


Рис. 23 — Візуальне представлення системи

Вимоги до лог файлу. Варіант 2

log-файл буде мати назву shootingLog, коли він був створений;

log-файл повинен створюватись, якщо він ще не існує;

• якщо log-файл існує, вміст буде записуватись в кінець;

• якщо log-файл досягає розміру 50 мб, буде створений поряд в тій ж папці новий log-файл з новим іменем за першим правилом;

• log-файл повинен мати розширення .log;

• кожен запис log-файлу повинен супроводжуватись часом, коли дія була викликана, її назвою (тип переривання) та інші додаткові деталі.

Завдання Варіант 3. Опис технологічного процесу

Система управління доступом в приміщення.

У даній системі буде два приміщення, у кожного з яких на вході розташовано по два турнікети, що управляються одним комп'ютером. Існують певні правила роботи системи турнікетів:

• За замовчуванням турнікети знаходяться у режимі очікування та заблоковані і на вхід, і на вихід.

• Турнікет не може бути одночасно розблокованим на вхід та на вихід.

• У разі одночасної спроби зчитування карток і на вході турнікету, і на виході, одна зі сторін сигналізує про помилку, а інша розблоковується для проходження. Пріоритет залежить від часу доби: до 12:00 вищий пріоритет у входу, а після 12:00 - у виходу.

• Увійти у приміщення з певною карткою можна лише тоді, якщо перед цим ця картка не була вже використана на вхід у будь-яке приміщення (людина знаходиться ззовні), або ж ще не була використана взагалі.

• Вийти з приміщення з певною карткою можна лише тоді, коли перед цим ця картка була використана для входу у це ж приміщення.

Для усіх чотирьох турнікетів алгоритми є однаковими та можуть виконуватися паралельно. Проте потік виконання у рамках одного турнікету

залежить від попередніх подій на цьому ж або інших турнікетах (повинна контролюватися логічна послідовність входів та виходів з приміщень за допомогою однієї картки).

Алгоритм системи. Варіант 3

Після запуску турнікету, і вхід, і вихід за замовчуванням переводять у режим очікування.

Алгоритми для варіантів використання:

ВВ1. Вхід у приміщення

1.1 Передумови: перебування входу турнікета у режимі очікування.

1.2 Головний потік:

• зчитуються дані картки на вході [А1];

• вхід розблоковується, індикатор ходу загоряється зеленим та очікується проходження у приміщення [А2], а на виході блокується можливість зчитування картки;

• вхід блокується, можливість зчитування картки на виході розблоковується;

• вхід турнікету переводять у режим очікування, індикатор ходу загоряється помаранчевим;

1.3 Альтернативні потоки.

[А1] Якщо картка на вході була зчитана успішно, то її дані відправляються у БД для фіксації зчитування, продовжується головний потік виконання. Якщо картку не було прочитано коректно або одночасно була зчитана картка на виході, генерується переривання. Якщо картка перед цим була використана на вхід у це ж або інше приміщення (тобто, на будь-якому вході), турнікет протягом 3 секунд сигналізує на вході про помилку червоним кольором індикатора, а потім вхід повертається у режим очікування.

[А2] Якщо протягом 10 секунд було зафіксоване проходження через турнікет у приміщення, то дані про це відправляються у БД для фіксації, а

турнікет на вході блокується, не очікуючи завершення інтервалу 10 секунд. Якщо ні, то турнікет блокується на вході після 10 секунд.

ВВ2. Вихід з приміщення

2.1 Передумови: перебування виходу турнікета у режимі очікування.

2.2 Головний потік:

• зчитуються дані картки на виході [А1];

• вихід розблоковується, індикатор виходу загоряється зеленим та очікується проходження з приміщення [А2], а на вході блокується можливість зчитування картки;

• вихід блокується, можливість зчитування картки на вході розблоковується;

• вихід турнікету переводиться у режим очікування, індикатор виходу загоряється помаранчевим;

2.3 Альтернативні потоки.

[А1] Якщо картка на виході була зчитана успішно, то її дані відправляються у БД для фіксації зчитування, продовжується головний потік виконання. Якщо картку не було прочитано коректно або одночасно була прочитана картка на вході, генерується переривання. Якщо картка перед цим була використана інакше, ніж на вхід у це приміщення, турнікет протягом 3 секунд сигналізує про помилку червоним кольором індикатора виходу, а потім вихід повертається у режим очікування.

[А2] Якщо протягом 10 секунд було зафіксоване проходження через турнікет з приміщення, то дані про це відправляються у БД для фіксації, а турнікет на виході блокується, не очікуючи завершення інтервалу 10 секунд. Якщо ні, то турнікет блокує вихід після 10 секунд.

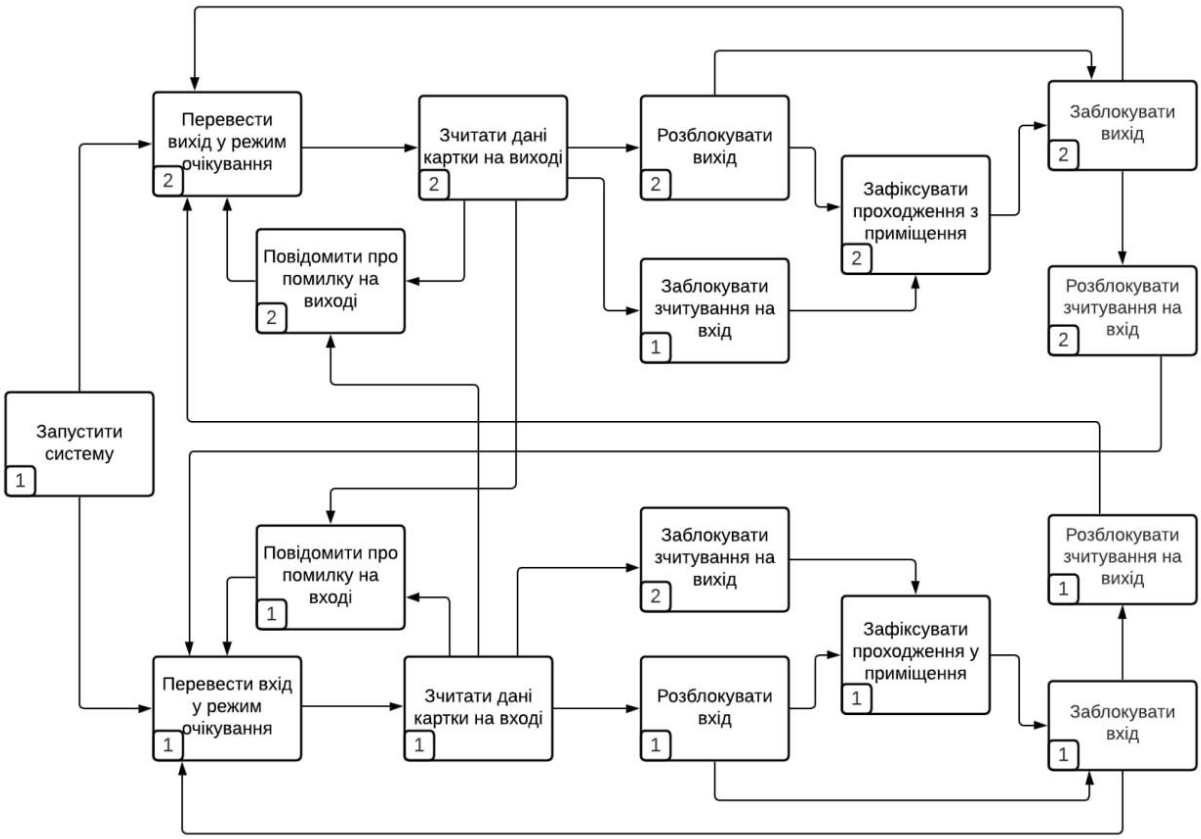


Рис. 24 — Паралельно – послідовний алгоритм системи управління доступом в приміщення

Список команд. Варіант 3

Список команд:

1. Запустити систему
2. Перевести вхід у режим очікування
3. Перевести вихід у режим очікування
4. Зчитати дані картки на вході
5. Зчитати дані картки на виході
6. Розблокувати вхід
7. Розблокувати вихід
8. Зафіксувати проходження у приміщення
9. Зафіксувати проходження з приміщення
10. Заблокувати вхід

11. Заблокувати вихід

12. Заблокувати зчитування на вхід

13. Заблокувати зчитування на вихід

14. Розблокувати зчитування на вхід

15. Розблокувати зчитування на вихід

16. Повідомити про помилку на вході

17. Повідомити про помилку на виході

Вимоги до візуалізації. Варіант 3

Програма представляє собою додаток, що зображає систему керування турнікетами у вигляді розподілених блоків. Елементи системи містять:

1.Зображення чотирьох турнікетів (по 2 у різних будівлях) та їх станів;

2.Індикатори та властивості кожного турнікету;

3.Панель керування станами турнікетів, що може ініціювати переривання, розблокування тощо.

Турнікети нумеруються згідно з розташуванням у будівлях.Турнікети можуть бути у декількох станах: у режимі очікування, у режимі підтвердження, у режимі помилки, а також у вимкненому режимі. Як і на індикаторах турнікетів ці стани зображуються відповідними кольорами: стан очікування - помаранчевим, підтвердження - зеленим, помилки - червоним, вимкнений стан - індикатор не зображує жодного кольорового забарвлення. Окрім цього, ведеться підрахунок людей, що зайшли до приміщення через турнікет.

Вимоги до лог файлу. Варіант 3

Log-file повинен створюватися, якщо він відсутній;

• Log-file матиме назву виду “ДД-ММ-РР”, де зазначено дату його створення;

• Для кожної нової доби повинен бути окремий Log-file;

• Log-file повинен мати розширення .txt;

• Log-file повинен відображати всі дії та процеси в системі в форматі: Час / тип / назва / деталі;

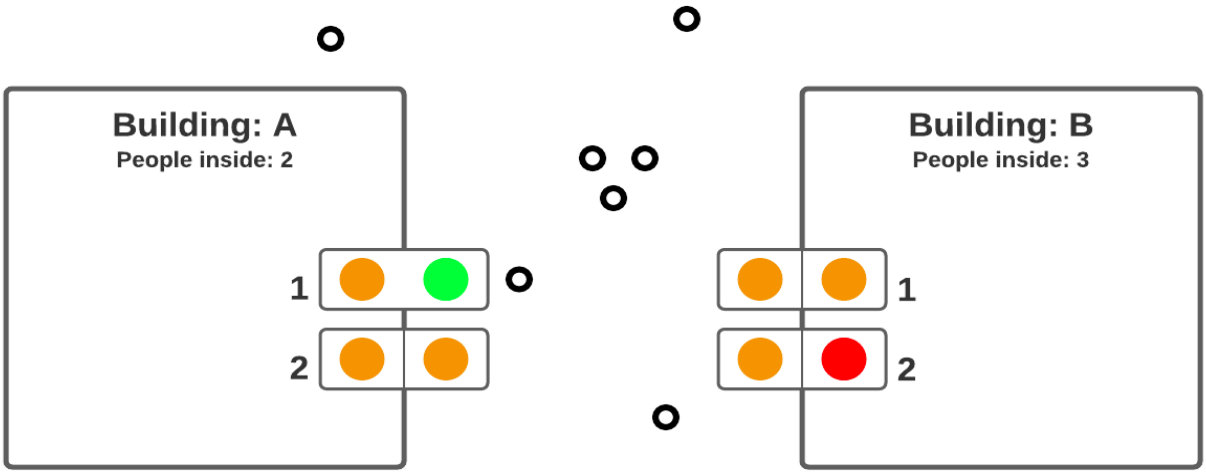


Рис. 25 — Візуальне представлення системи доступу до приміщення.

Програма проведення експерименту ЛР 8

Студент обирає одну з запропонованих тем. Студент самостійно обирає мову програмування та методи забезпечення паралельного виконання програми. При цьому потрібно передбачити наступні кроки:

- провести первинний аналіз завдання, створення технічного завдання на розробку системи (постановка задачі, послідовно-паралельний алгоритм системи, вимоги до візуалізації, вимоги до лог файлу);
- програмування системи відповідно до вимог технічного завдання;
- тестування програмної системи;
- оформлення звіту до лабораторної роботи відповідно до наданих дали вимог.

Теоретичні відомості ЛР 8

Стан процесу. Керування процесами.

Керування процесом включає виконання деяких операцій над ним, дозволяючи процесу пройти визначені стани. Види станів процесу

визначаються конкретною ОС і системою керування процесами, але, містять здебільшого такі стани:

- Породження – створення процесу і підготовка до першого виконання в системі;
- Готовності – процес готовий до використання і чекає звільнення процесор;
- Виконання – процес виконується на процесорі;
- Блокування – процес призупинений через очікування визначеної події: завершення введення даних, завершення заданого часу очікування, сигналу від іншого процесу, звільнення ресурсу та ін.
- Завершення – нормальне або аварійне завершення виконання процесу.

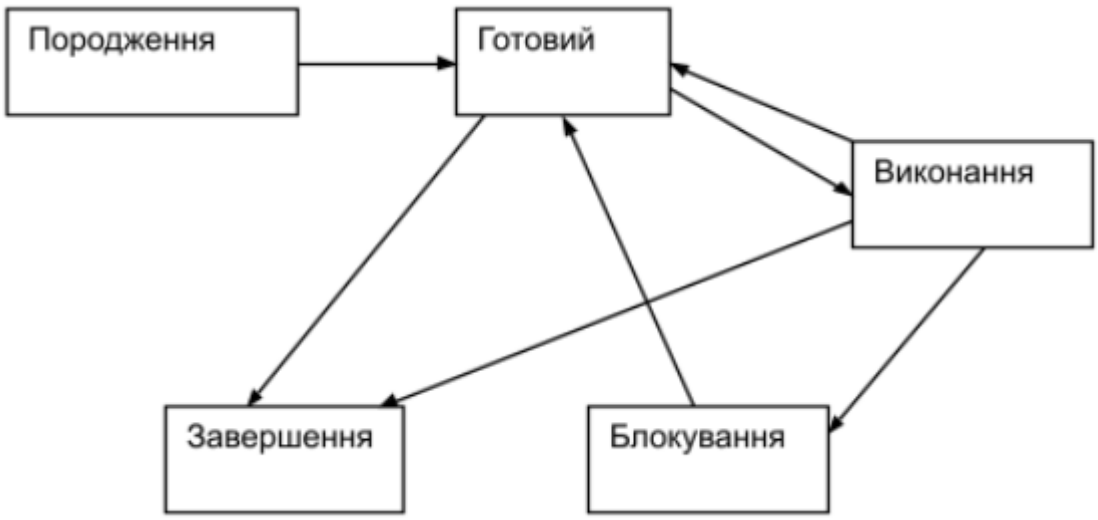


Рис. 26 — Діаграма станів процесу

Процес також може мати особливий стан, який отримав назву тупик (deadlock). Процес у тупиковому стані блокований і очікує на подію, яка ніколи не відбудеться. Це зумовлює неможливість його продовження і, як наслідок, - зависання програми в цілому. Тупики – одна з головних проблем, що виникають під час виконання паралельних програм. Боротьба з тупиками зводиться до таких дій:

- відвернення тупиків;

- запобігання тупикам;
- визначення тупика;
- ліквідація тупика.

Засоби роботи з процесами мають забезпечити розробнику паралельного додатка такі можливості:

- оголосити процес (групу процесів);
- встановити пріоритет процесу;
- запустити процес на виконання;
- призупинити процес на визначений час;
- блокувати і розблокувати процес;
- організувати взаємодію з іншими процесами (синхронізація або передавання даних);
- завершити процес.

З погляду паралельних властивостей усі задачі можна поділити на три групи:

- Повністю паралельні задачі;
- Частково паралельні задачі;
- Задачі, які не допускають паралельної обробки.

Прикладом повністю паралельної задачі є операція додавання матриць, яку можна виконати за один крок, на якому здійснюється формування кожного елемента матриці.

Частково паралельною є задача, у якій чергуються паралельні та часткові частини. Приклад такої задачі - операція скалярного множення векторів.

Існують також задачі, для яких неможливо побудувати паралельний алгоритм. Наприклад, операція a=b+c+d не може бути подано у вигляді паралельного алгоритму.

Розроблення паралельного алгоритму складається з окремих кроків:

– Декомпозиція

– Проектування взаємодії задач

– Об’єднання

– Планування обчислень

Декомпозиція

Існують різні види декомпозиції: декомпозиція за даними, функціональна декомпозиція, об’єктна декомпозиція та ін. Кількість під задач має перевищувати кількість процесорів комп’ютерної системи (КС), що дозволить оптимізувати наступні етапи розроблення алгоритму. При цьому слід мінімізувати кількість обчислень, а також обсяг інформації, що пересилається між під задачами.

Проектування взаємодії підзадач

Визначаються дії і методи, які потрібні для організації взаємодії під задач (передавання даних, синхронізація, доступ до спільних даних та ін.). Види взаємодії можуть бути локальними чи глобальними залежно від того, яку кількість зв’язків з іншими має кожна під задача;синхронними чи асинхронними залежно від виду виконання обміну даних; статичними чи динамічними, якщо вони можуть змінюватися в процесі виконання програми.

Об’єднання

Виконується об’єднання окремих під задач у більші задачі з метою зменшення обсягів даних, які потрібно передавати між задачами. Кінцева мета об’єднання – збільшення ефективності алгоритму за рахунок скорочення часу його виконання і складності реалізації.

Планування обчислень

Виконується розподіл задач по процесорах КС. Оптимальне розміщення дозволить мінімізувати реальні затрати часу, які будуть пов’язані з виконанням обчислень і обміном даних між процесорами системи.

1. Декомпозиція

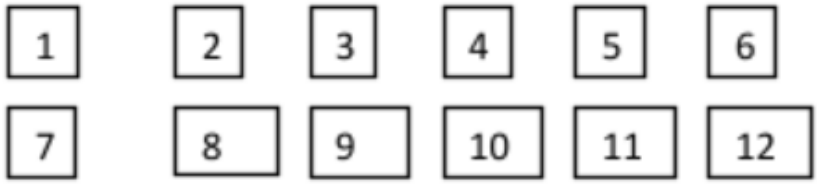


Рис. 27 — Декомпозиція

2. Взаємодія підзадач

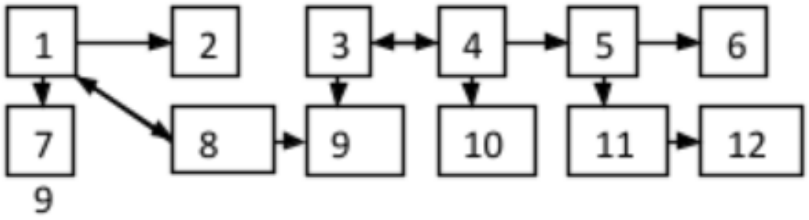


Рис. 28 — Взаємодія підзадач

3. Об'єднання

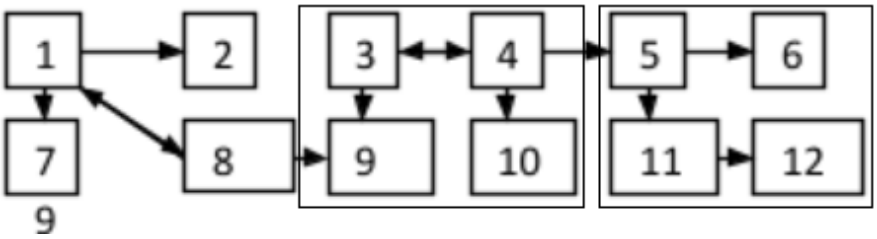


Рис. 29 — Об'єднання

4. Планування обчислень

Рис. 30 — Планування обчислень

Будемо розглядати тільки послідовно-паралельні процеси, тобто такі процеси, які можуть бути розбиті тільки на послідовні та паралельні ланки.
Наведемо приклад послідовно-паралельного графу (рис. 31):
P1;P2||P3;P4;P5||P6;P7;P8;P9

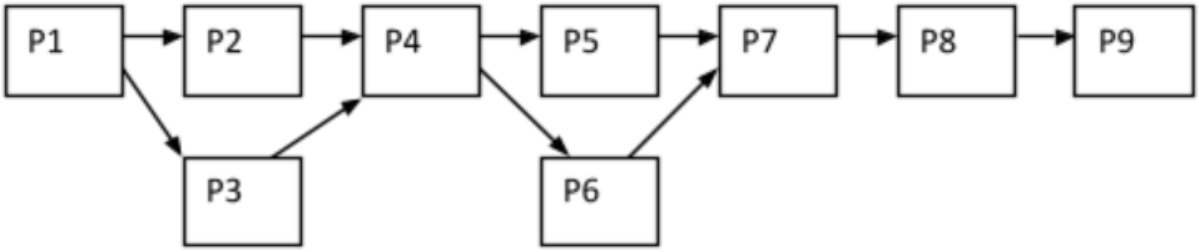


Рис. 31 — Послідовно-паралельний граф

Існує два основні засоби опису послідовно-паралельних процесів.

1) використання спеціального символу, що поєднує процеси, які виконуються паралельно;

2) використання спеціальних операторних скобок для об'єднання паралельних процесів.

Наприклад, в першому випадку можна використовувати символ«;» (крапка з комою) покажчиком послідовного виконання процесу, а символ «||»– покажчиком паралельного виконання процесів.

Семафори

Механізм семафорів- це спеціальний захищений тип Semaphore та дві неподільні операції над змінною S цього типу: P(S) і V(S). Неподільність операції означає, що її не можна переривати, поки не завершиться її виконання. Бінарні семафори набувають значень 0 і 1, багатозначні (множні) семафори – значень 0,1,… M.

Алгоритм операцій P(S) і V(S)

Операція P(S) (ВХІДКД)
Перевірити значення S.

Якщо S = 0, то блокувати процес, що виконує цю операцію.

Інакше S= S-1 і дозволити входження процесу в критичну ділянку.

Операція V(S) (ВИХІДКД)

S= S+1

Механізм семафорів є універсальним, який може бути використаний як для вирішення завдання взаємного виключення (рис. 32).

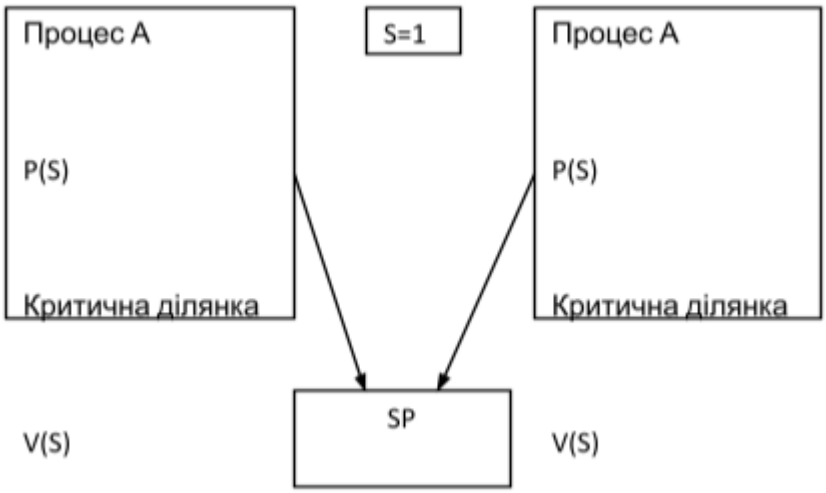


Рис. 32 — Загальна схема вирішення завдання взаємного виключення з використанням семафорів

Монітори

Монітор – програмний модуль, що містить змінні та процедури для роботи з ними, причому доступ до змінних можливий тільки через процедури монітора. Монітор – засіб розподілу ресурсів і взаємодії процесів. Це призначення монітора реалізується за допомогою властивостей, якими наділені процедури монітора. Характерна особливість процедур монітора – взаємне виключення ними одне одного. У будь-який момент часу може виконуватися тільки одна процедура монітора. Якщо будь-який процес викликав і виконує процедуру монітора, то жоден процес не може виконувати будь-які процедури, що виконується, або іншої процедури монітора цей процес блокується і розміщується в черзі блокованих процесів доти, доки активний процес не закінчить виконання процедури монітора. Тобто в моніторі не може

«Знаходитись» більше одного процесу. Така властивість процедур монітора забезпечує взаємне виключення процесів, які працюють з монітором.
Загальна структура монітора:

Monitor ім’я монітора;
Опис локальних даних
Опис процедур для доступу до даних
Begin
Ініціалізація локальних даних
End ім’я монітора;

9.1. Контрольні запитання до лабораторної роботи № 8

1. Що включає в себе структура паралельної КС?
2. Які існують види організації пам’яті?
3. Якими топологічними параметрами характеризується структура комп’ютерної системи з розподіленою пам’яттю?
4. Надати визначення процесу?
5. Які бувають стани процесів?
6. Які методи використовуються для боротьби з тупиками?
7. Які завдання треба вирішувати для вирішення задачі взаємодії процесів?

10. ОФОРМЛЕННЯ ЗВІТУ ТА ПОРЯДОК ЙОГО ПОДАННЯ

Звіт можна подавати в електронній формі та у роздрукованому вигляді. Затримка подання роздрукованого звіту не повинна перевищувати дві лабораторні роботи. У випадку більшої затримки можуть нараховуватися штрафні бали. Під час захисту роботи необхідною умовою захисту лабораторної роботи є]:

– наявність файлу з *вихідним кодом*, що не містить синтаксичних помилок і помилок на етапі виконання програми;

– у вихідному коді коментарі з прізвищем, номером групи, номером лабораторної роботи;

– наявність блок-схеми алгоритмів, кількість лабораторних робіт з блок-схемою має визначатися викладачем;

– здатність студента пояснити кожний рядок вихідного коду;

– здатність студента зробити зміни у програмі і налагодити після цього програму;

– здатність студента здійснити покроковий запуск програми під час налаштування, при цьому студент повинен пояснити змінні в пам’яті програми під час кожного кроку; пояснити призначення кожної інструкції. У разі необхідності до програми додаються фотографії екранів (скріншот) і відповідні обчислення, що обумовлено у кожній лабораторної роботі.

Код повинен містити необхідні коментарі, його дозволено використовувати під час захисту роботи. У випадку помилок на етапі компіляції програми або під час її запуску робота до захисту не допускається.

У звіті має бути висновок, що містить інформацію про ускладнення, що виникли під час розробки і налагодження програми, відповідність результатів теоретичним положенням.

Титульний аркуш для звіту подано далі.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАЗВА ФАКУЛЬТЕТУ
НАЗВА КАФЕДРИ

ЛАБОРАТОРНА РОБОТА № 1

З ДИСЦИПЛІНИ «Інформатика.

Основи програмування та алгоритми»

Виконав:

студент групи РТ – 00

_____

«__»_____

Перевірив:

_____

«__»_____

Київ – 202_

СПИСОК ЛІТЕРАТУРИ

Основна література

1. Алгоритми та методи обчислень [Електронний ресурс] : навч. посіб. для студ. спеціальностей 121 «Інженерія програмного забезпечення», спеціалізації «Програмне забезпечення високопродуктивних комп'ютерних систем та мереж» та 123 «Комп'ютерна інженерія», спеціалізації «Комп'ютерні системи та мережі» / М. А. Новотарський; КПІ ім. Ігоря Сікорського. — Електронні текстові дані (1 файл: 4648 Кбайт). — Київ : КПІ ім. Ігоря Сікорського, 2019. — 407 с. Microsoft Word — Алгоритми та методи обчислень Новотарський.docx (kpi.ua)

2. Базові концепції програмування [електронний ресурс]: Навчальний посібник до виконання комп'ютерного практикуму для студентів спеціальності 122 — «Комп'ютерні науки», освітньо-професійної програми «Комп'ютерний моніторинг та геометричне моделювання процесів і систем» / Укладач: Крячок О.С. — електронні текстові дані (1 файл: 885 КБайт). — К.: КПІ ім. Ігоря Сікорського, 2021. — 65 с.

3. Керніган Б.В., Річі Д.М. Мова програмування С. / Б. В. Керніган, Д. М. Річі — К.: Знання, 2015. — 304 с.

4. Катін П. Ю. Архітектура комп'ютера : лаб. практикум [Електронний ресурс]: навальний. посібник для студ. спец. 121 «Інженерія програмного забезпечення» / П. Ю. Катін. — Електронні текстові дані (1 файл: 2,2 Мбайт). — Київ: КПІ ім. Ігоря Сікорського, 2021. — 123 с.ELAKPI: Архітектура комп'ютера. Лабораторний практикум (https://ela.kpi.ua/handle/123456789/45717)

5. Шаповаленко В. А. Чисельне обчислення функцій, характеристик матриць і розв'язування нелінійних рівнянь та систем рівнянь: Навч.

посібник / В.А.Шаповаленко, Л.М. Буката, О.Г Трофименко. — Одеса: ВЦ ОНАЗ, 2010. — Ч. 1. — 88 с.

6. Habrison S. C: A Reference Manual / S. Habrison — Pearson, 2002. — 533.

7. Deitel P.J., Deitel H.M. C: How to Program / P.J. Deitel, H.M. Deitel. — Pearsons, 2015. — 1008 p.

8. Лінійна алгебра та аналітична геометрія. Частина 1 [Електронний ресурс] : навч. посіб. для студ. спеціальності 113 «Прикладна математика», спеціалізації «Data Science та математичне моделювання» / В.В. Мальчиков, В.В. Третиник, К.О. Костенко; КПІ ім. Ігоря Сікорського. — Електронні текстові дані (1 файл: 4, 289 Мбайт). — Київ : КПІ ім. Ігоря Сікорського, 2020. — 194 с.

9. Татарчук Д. Д., Діденко Ю. В. Програмування мовами С та С++: навч. посіб. / Д.Д. Татарчук, Ю.В. Діденко. — Київ, 2012. — 112 с.

10. Ivanov N. Программирование в Linux с нуля / N. Ivanov — Режим доступу:https://web.archive.org/web/20070630101342/http://www.lindevel.ru/zlp — Дата доступу: 01.12.21. — Назва з екрану.

11. Системи реального часу [Електронний ресурс] : методичні вказівки до виконання лабораторних робіт для студентів напряму підготовки 6.050102 «Комп’ютерна інженерія» кафедри обчислювальної техніки всіх форм навчання / НТУУ «КПІ»; уклад. В. Є. Мухін, А. М. Волокита, Я. І. Корнага ; відп. ред. В. П. Широчин. — Електронні текстові дані (1 файл: 487 Кбайт). – Київ : НТУУ «КПІ», 2013. - 40 с. — Назва з екрана. https://ela.kpi.ua/handle/123456789/4826

Додаткова література

1. Шпак З.Я. Програмування мовою С: Навчальний посібник / З.Я. Шпак. — Оріяна-Нова, 2006. — 432.

2. Вінник В.Ю. Алгоритмічні мови та основи програмування: навч. посібник / В.Ю. Вінник — Ж: ЖДТУ, 2007. — 328 с.

3. Kernighan B.W., Ritchie D.M. C Programming Language. / B.W. Kernighan, D.M. Ritchie. — Prentice Hall Software Series., 1988 — 288 p.

4. Schildt H. C: The Complete Reference / H. Schildt. — McGraw Hill, 2000. — 805 p.

5. Prata S. C Primer Plus / S. Prata. — Pearsons Education, 2004. — 984 p.

6. Tanenbaum A. Structured Computer Organization / A. Tanenbaum. — Pearson, 2016. — 1080 p.

7. Deitel P., Deitel H. C++: How to Program / P. Deitel, H. Deitel. — Pearson, 2015. — 1008 p.

8. Sedgewick R. Algorithms in C++ / R. Sedgewick. — Addison-Weasley, 1998. — 752 p.

9. Schildt H. C/C++ Programmer's Reference / H. Schildt. — McGraw-Hill Professional, 2000. — 416 p.

ДОДАТОК 1. ДОМАШНЯ КОНТРОЛЬНА РОБОТА

Для того, щоб ДКР була зарахованою необхідно виконати завдання № 1 та № 2. Завдання № 3 — не обов'язкове і виконується студентами, які бажають мати змогу отримати максимальний бал за ДКР.

Номер варіанту визначається порядковим номером в списку групи.

Завдання № 1

1.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = \sum_{n=0}^{N} \frac{a^n}{n!}$, значення змінних a та N $(N \geq 0)$ вводяться з клавіатури. N - ціле число.
2.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = 1 + \sum_{n=1}^{N} (-1)^{(n-1)} \frac{a^n}{n!}$, значення змінних a та N $(N \geq 1)$ вводяться з клавіатури. N - ціле число.
3.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = \sum_{n=0}^{N} (-1)^n \frac{a^n}{n!}$, значення змінних a та N $(N \geq 0)$ вводяться з клавіатури. N - ціле число.
4.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = 1 + \sum_{n=1}^{N} \frac{a^n}{a+n}$, значення змінних a, $(a \geq 0)$ та N $(N \geq 1)$ вводяться з клавіатури. N - ціле число.

5.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = 1 + \sum_{n=1}^{N} (-1)^{(n-1)} \frac{a^n}{a+n}$, значення змінних a, $(a \geq 0)$ та N $(N \geq 1)$ вводяться з клавіатури. N - ціле число.
6.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = 1 + \sum_{n=1}^{N} (-1)^{(n-1)} \frac{a^n}{a+(-1)^n n}$, значення змінних a та N $(N \geq 1)$ вводяться з клавіатури. N - ціле число. Якщо для певного члена ряду знаменник $a+(-1)^n n$ виявляється рівним нулю, тоді такий член відкидається і в підрахунку суми не використовується.
7.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = 1 + \sum_{n=1}^{N} (-1)^{n} \frac{a^n}{a+n}$, значення змінних a та N $(N \geq 1)$ вводяться з клавіатури. N - ціле число. Якщо для певного члена ряду знаменник $a+n$ виявляється рівним нулю, тоді такий член відкидається і в підрахунку суми не використовується.
8.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = 1 + \sum_{n=1}^{N} (-1)^{n} \frac{a^n}{a+(-1)^n n}$, значення змінних a та N $(N \geq 1)$ вводяться з клавіатури. N - ціле число. Якщо для певного члена ряду знаменник $a+(-1)^n n$ виявляється рівним нулю, тоді такий член відкидається і в підрахунку суми не використовується.
9.	Не використовуючи стандартні математичні функції, розрахувати суму ряду.

	$S = 1 + \sum_{n=1}^{N} (-1)^n \frac{a^n}{a + (-1)^{(n-1)} n}$, значення змінних a та N $(N \geq 1)$ вводяться з клавіатури. N - ціле число. Якщо для певного члена ряду знаменник $a + (-1)^{(n-1)} n$ виявляється рівним нулю, тоді такий член відкидається і в підрахунку суми не використовується.
10.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = 1 + \sum_{n=1}^{N} (-1)^{(n-1)} \frac{a^n}{a + (-1)^{(n-1)} n}$, значення змінних a та N $(N \geq 1)$ вводяться з клавіатури. N - ціле число. Якщо для певного члена ряду знаменник $a + (-1)^{(n-1)} n$ виявляється рівним нулю, тоді такий член відкидається і в підрахунку суми не використовується.
11.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = \sum_{n=0}^{N} (-1)^n \frac{a^{2n+1}}{(2n+1)!}$, значення змінних a та N $(N \geq 0)$ вводяться з клавіатури. N - ціле число.
12.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = \sum_{n=0}^{N} \frac{a^{2n+1}}{(2n+1)!}$, значення змінних a та N $(N \geq 0)$ вводяться з клавіатури. N - ціле число.
13.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = \sum_{n=0}^{N} (-1)^n \frac{a^{2n}}{(2n)!}$, значення змінних a та N $(N \geq 0)$ вводяться з

	клавіатури. N - ціле число.
14.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S=\sum_{n=0}^{N}\frac{a^{2n}}{(2n)!}$, значення змінних a та N $(N\geq 0)$ вводяться з клавіатури. N - ціле число.
15.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S=\sum_{n=1}^{N}(-1)^{n+1}\frac{a^n}{n}$, значення змінних a та N $(N\geq 1)$ вводяться з клавіатури. N - ціле число.
16.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S=\sum_{n=1}^{N}\frac{a^n}{n}$, значення змінних a та N $(N\geq 1)$ вводяться з клавіатури. N - ціле число.
17.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S=\sum_{n=1}^{N}\frac{1}{n}$, значення змінної N $(N\geq 1)$ вводиться з клавіатури. N - ціле число.
18.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S=\sum_{n=1}^{N}(-1)^{n-1}\frac{1}{n}$, значення змінної N $(N\geq 1)$ вводиться з клавіатури. N - ціле число.
19.	Не використовуючи стандартні математичні функції, розрахувати суму ряду.

	$S = 2 + \sum_{n=1}^{N} (-1)^{n-1} \frac{n}{n+1}$, значення змінної N $(N \geq 1)$ вводиться з клавіатури. N - ціле число.
20.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = \sum_{n=1}^{N} \left(2^n - 2^{n-1}\right)$, значення змінної N $(N \geq 1)$ вводиться з клавіатури. N - ціле число.
21.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = \sum_{n=1}^{N} \left(\frac{2}{n} + \frac{n}{2}\right)$, значення змінної N $(N \geq 1)$ вводиться з клавіатури. N - ціле число.
22.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = \sum_{n=1}^{N} \left(n + \frac{n}{2}\right)$, значення змінної N $(N \geq 1)$ вводиться з клавіатури. N - ціле число.
23.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = \sum_{n=0}^{N} (-1)^n \frac{2^n}{n!}$, значення змінної N $(N \geq 0)$ вводяться з клавіатури. N - ціле число.
24.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = 1 + \sum_{n=0}^{N} \frac{2^n}{n!}$, значення змінної N $(N \geq 0)$ вводяться з клавіатури. N - ціле число.
25.	Не використовуючи стандартні математичні функції, розрахувати суму

	ряду. $S = a^{N-1} + \sum_{n=1}^{N} n$, значення змінних a та N $(N \geq 0)$ вводяться з клавіатури. N - ціле число.
26.	Не використовуючи стандартні математичні функції, розрахувати суму ряду. $S = \frac{a^{N-1}}{3} + \sum_{n=1}^{N} \frac{n}{2}$, значення змінних a та N $(N \geq 0)$ вводяться з клавіатури. N - ціле число.

Завдання № 2

1.	Написати програму, яка підраховує кількість елементів в одномірному статичному масиві цілих чисел, що кратні 5. Алгоритм реалізувати у вигляді окремої функції, яка отримує два аргументи — масив та розмір масиву, і повертає знайдене значення.
2.	Написати програму, яка заходить суму від'ємних елементів в одномірному статичному масиві. Алгоритм реалізувати у вигляді окремої функції, яка отримує два аргументи — масив та розмір масиву, і повертає знайдене значення.
3.	Написати програму, яка сортує за спаданням (від більшого до меншого) елементи одномірного статичного масиву. Алгоритм реалізувати у вигляді окремої функції типу void, яка отримує два аргументи - масив та розмір масиву. Виведення елементів одномірного статичного масиву на екран реалізувати у вигляді окремої функції.
4.	Написати програму,
5.	Задана квадратна матриця A розміром $N \times N$. Значення елементів матриці A вводяться з клавіатури. Задається матриця B розміром

	$N \times N$. Виконати сортування за зростанням стовбців матриці A, результат зберегти в матриці B. (Матриця B зберігає результат, при цьому матриця A зберігається в початковому вигляді). Вивести на екран матриці A та B. Задачу реалізувати у вигляді окремої функції.
6.	Задана квадратна матриця A розміром $N \times N$. Значення елементів матриці A вводяться з клавіатури. Задається матриця B розміром $N \times N$. Виконати сортування рядків матриці за зростанням. Результат сортування зберегти в матрицю B. Після цього виконати сортування стовбців матриці B за зростанням. (Матриця B зберігає результат, при цьому матриця A зберігається в початковому вигляді). Вивести на екран матриці A та B. Задачу реалізувати у вигляді окремої функції.
7.	Задана квадратна матриця A розміром $N \times N$. Значення елементів матриці A вводяться з клавіатури. Задається матриця B розміром $N \times N$. Виконати впорядкування рядків матриці по спаданню значень сум елементів рядків. Результат сортування зберегти в матрицю B. (Матриця B зберігає результат, при цьому матриця A зберігається в початковому вигляді). Вивести на екран матриці A та B. Задачу реалізувати у вигляді окремої функції.
8.	Задана квадратна матриця A розміром $N \times N$. Значення елементів матриці A вводяться з клавіатури. Знайти добуток елементів матриці, що знаходяться на головній діагоналі. Вивести на екран матрицю А та розраховане значення. Задачу реалізувати у вигляді окремої функції.
9.	Задана квадратна матриця A розміром $N \times N$. Значення елементів матриці A вводяться з клавіатури. Розрахувати суму елементів матриці A, що знаходяться вище головної діагоналі (включаючи головну діагональ). Вивести на екран матрицю А та розраховане значення. Задачу реалізувати у вигляді окремої функції.
10.	Задана квадратна матриця A розміром $N \times N$. Значення елементів

	матриці A вводяться з клавіатури. Знайти найбільше значення серед елементів, що знаходяться нижче головної діагоналі матриці (включаючи головну діагональ). Вивести на екран матрицю A, знайдене значення та індекси елемента. Задачу реалізувати у вигляді окремої функції.
11.	Задана квадратна матриця A розміром $N \times N$. Матриця A заповнюється випадковими числами із діапазону $-a...a$, де a - випадкове число із діапазону $1...10$. Задається матриця B розміром $N \times N$. Елементи матриці B формуються як сума елементів матриці A без врахування елемента,який відповідає поточному елементу матриці B. Наприклад, елемент матриці B з індексами [0][0] буде розраховуватися як сума елементів матриці A за виключення елемента $A[0][0]$. Елемент $B[0][1]$ розраховується як сума елементів матриці A за виключення елемента $A[0][1]$ і т.д.
12.	Задана квадратна матриця A розміром $N \times N$. Матриця A заповнюється випадковими числами із діапазону $-a...a$, де a - випадкове число із діапазону $5...9$. Задається матриця B розміром $N \times N$. Виконати сортування за зростанням стовбців матриці A, результат зберегти в матриці B. (Матриця B зберігає результат, при цьому матриця A зберігається в початковому вигляді). Вивести на екран матриці A та B. Задачу реалізувати у вигляді окремої функції.
13.	Задана квадратна матриця A розміром $N \times N$. Матриця A заповнюється випадковими числами із діапазону $0...a$, де a - випадкове число із діапазону $10...15$. Задається матриця B розміром $N \times N$. Виконати сортування за спаданням елементів матриці A, що знаходяться на головній діагоналі. Результат зберегти в матриці B. (Матриця B зберігає результат, при цьому матриця A зберігається в початковому вигляді). Вивести на екран матриці A та B. Задачу

	реалізувати у вигляді окремої функції.
14.	Задана квадратна матриця A розміром $N \times N$. Матриця A заповнюється випадковими числами із діапазону $-a...a$, де a - випадкове число із діапазону $5...10$. Визначити індекс стовбця, елементи якого формують найменше значення суми. Вивести на екран матрицю A, індекс стовбця та розраховану суму. Задачу реалізувати у вигляді окремої функції.
15.	Задана квадратна матриця A розміром $N \times N$. Матриця A заповнюється випадковими числами із діапазону $-a...a$, де a - випадкове число із діапазону $30...55$. Задається матриця B розміром $N \times N$. Виконати сортування рядків матриці за зростанням. Результат сортування зберегти в матрицю B. Після цього виконати сортування стовбців матриці B за зростанням. (Матриця B зберігає результат, при цьому матриця A зберігається в початковому вигляді). Вивести на екран матриці A та B. Задачу реалізувати у вигляді окремої функції.
16.	Задана квадратна матриця A розміром $N \times N$. Матриця A заповнюється випадковими числами із діапазону $-a...a$, де a - випадкове число із діапазону $3...7$. Задається матриця B розміром $N \times N$. Виконати впорядкування рядків матриці по зростанню значень сум елементів рядків. Результат сортування зберегти в матрицю B. (Матриця B зберігає результат, при цьому матриця A зберігається в початковому вигляді). Вивести на екран матриці A та B. Задачу реалізувати у вигляді окремої функції.
17.	Задана квадратна матриця A розміром $N \times N$. Матриця A заповнюється випадковими числами із діапазону $-a...a$, де a - випадкове число із діапазону $5...12$. Знайти добуток елементів матриці,

	що знаходяться на головній діагоналі. Вивести на екран матрицю A та розраховане значення. Задачу реалізувати у вигляді окремої функції.
18.	Задана квадратна матриця A розміром $N \times N$. Матриця A заповнюється випадковими числами із діапазону $-a...a$, де a - випадкове число із діапазону $4...8$. Розрахувати добуток елементів матриці A, що знаходяться вище головної діагоналі (включаючи головну діагональ). Вивести на екран матрицю A та розраховане значення. Задачу реалізувати у вигляді окремої функції.
19.	Написати програму, яка заходить кількість елементів в одномірному статичному масиві, до дорівнюють одному з наступних значень: 2, 5, 7, 10. Тип даних масиву — unsigned int. Значення елементів масиву задаються в тексті програми. Алгоритм реалізувати у вигляді окремої функції, яка отримує два аргументи — масив та розмір масиву, і повертає знайдене значення, яке виводиться на екран монітору. Наприклад, якщо задано масив, який складається із таких елементів: 4 12 **5** 8 **2** **10** 24 91 то шукане значення — 3
20.	Написати програму, яка заходить найменше по модулю від'ємне значення в одномірному статичному масиві. Тип даних елементів масиву — double. Значення елементів масиву вводяться з клавіатури. Алгоритм знаходження результату реалізувати у вигляді окремої функції, яка отримує два аргументи — масив та розмір масиву, і повертає знайдене значення, яке виводиться на екран монітору.
21.	Написати програму, яка знаходить значення, що найчастіше зустрічається в одномірному статичному масиві. Тип даних елементів

	масиву — int. Значення елементів масиву задаються з клавіатури. Можливий діапазон значень елементів масиву □□3;3□. На етапі введення значень елементів масиву забезпечити контроль щодо відповідності введеного значення заданому проміжку □□3;3□. Якщо введене значення елементу масиву виходить за заданий проміжок — забезпечити можливість користувачу виконати повторне введення. Алгоритм пошуку значення, що найчастіше зустрічається в масиві, реалізувати у вигляді окремої функції, яка отримує два аргументи — масив та розмір масиву, і повертає знайдене значення, яке виводиться на екран монітору.
22.	Написати програму, яка знаходить мінімальне значення в одномірному статичному масиві, а також індекс елементу, що має мінімальне значення. Знайдені результати виводяться на екран. Тип даних елементів масиві — double. Значення елементів масиву вводяться з клавіатури. Алгоритм реалізувати у вигляді окремої функції типу void. Функція отримує масив та розмір масиву, виконує пошук мінімального значення в масиві та його індекс, і виводить результати на екран.
23.	Написати програму, яка заходить суму елементів в одномірному статичному масиві. Тип даних елементів масиву — float Значення елементів масиву задаються в тексті програми. Алгоритм реалізувати у вигляді окремої функції, яка отримує два аргументи — масив та розмір масиву, і повертає знайдене значення, яке виводиться на екран монітору.
24.	Написати програму, яка виконує модифікацію елементів одномірного статичного масиву наступним чином: від'ємні значення замінюються на –1 додатні значення

	замінюються на 1 нульові значення залишаються без змін. Тип даних елементів масиву — int Алгоритм реалізувати у вигляді окремої функції типу void, яка отримує два аргументи — масив та розмір масиву. Також написати окрему функцію типу void яка забезпечує виведення елементів масиву на екран. Наприклад, якщо вхідний масив має значення: 4 0 –9 –2 8 12 –15 –3 0 7 то після модифікації, масив повинен містити наступні значення: 1 0 –1 –1 1 1 –1 –1 0 1
25.	Написати програму, яка підраховує добуток непарних значень в одномірному статичному масиві цілих чисел. Алгоритм реалізувати у вигляді окремої функції, яка отримує два аргументи — масив та розмір масиву, і повертає знайдене значення, яке виводиться на екран монітору.
26.	Написати програму, яка заходить найбільше додатнє значення в одномірному статичному масиві. Тип даних елементів масиву — double. Значення елементів масиву вводяться з клавіатури. Алгоритм реалізувати у вигляді окремої функції, яка отримує два аргументи — масив та розмір масиву, і повертає знайдене значення, яке виводиться на екран монітору.

Завдання № 3

1.	Написати програму, яка підраховує кількість слів **введених із клавіатури.** Програма виводить запит на введення тексту із клавіатури. Користувач вводить текст. Введення завершується натисканням комбінації клавіш Cntrl+Z, при цьому генерується значення EOF, що вказує на завершення введення текстової інформації. Текст, що вводиться з клавіатури складається із символів латинського алфавіту, цифрових символів, знаків пунктуації, пробільних символів. Підрахувати скільки слів було введено користувачем, при цьому окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації, однак набір лише символів пунктуації, які обмежені пробільними символами, словом не вважаються. Наприклад, якщо користувач ввів з клавіатури наступну послідовність: red! 445 black ...!!! [Cntrl+Z], то програма повинна розпізнати 3 слова. Результат підрахунку кількості слів **записується в текстовий файл.**
2.	Написати програму, яка знаходить **в текстовому файлі** (файл з розширенням *.txt) найдовше слово. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації, однак набір лише символів пунктуації, які обмежені пробільними символами, словом не вважаються. Для створення текстового файлу користуватися редактором Блокнот. Знайдене найдовше слово **виводиться на екран.**
3.	Написати програму, яка підраховує кількість символів, з яких

	складається найдовше слово в тексті, що **вводиться із клавіатури.** Користувач вводить текст. Введення завершується натисканням комбінації клавіш Cntrl+Z, при цьому генерується значення EOF, що вказує на завершення введення текстової інформації. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації, однак набір лише символів пунктуації, які обмежені пробільними символами, словом не вважаються. Програма **записує в текстовий файл** кількість символів, з яких складається найдовше слово.
4.	Написати програму, яка підраховує кількість слів, що складаються від 3 до 7 літер, в тексті, що **вводиться із клавіатури**. Користувач вводить текст. Введення завершується натисканням комбінації клавіш Cntrl+Z, при цьому генерується значення EOF, що вказує на завершення введення текстової інформації. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації, однак набір лише символів пунктуації, які обмежені пробільними символами, словом не вважаються. Програма **записує в текстовий файл** знайдену кількість слів. що складаються із кількості літер від 3 до 7.
5.	Написати програму, яка підраховує кількість цифр, що зустрічаються в тексті. Користувач вводить **текст з клавіатури**. Введення завершується натисканням комбінації клавіш Cntrl+Z, при цьому генерується значення EOF, що вказує на завершення введення текстової інформації. Окремим словом вважається набір символів, що не містить пробільних

	символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації. Програма **записує в текстовий файл** знайдену кількість цифрових символів.
6.	Написати програму, яка підраховує кількість знаків пунктуації, що зустрічаються в тексті. Знаками пунктуації вважати наступні символи: знак оклику, кома, крапка, тире, знак питання, двокрапка, крапка з комою. Користувач **вводить текст з клавіатури**. Введення завершується натисканням комбінації клавіш Cntrl+Z, при цьому генерується значення EOF, що вказує на завершення введення текстової інформації. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації. Програма **записує в текстовий файл** знайдену кількість знаків пунктуації.
7.	Написати програму, яка підраховує кількість цілих чисел, що зустрічаються в тексті. Користувач **вводить текст з клавіатури**. Введення завершується натисканням комбінації клавіш Cntrl+Z, при цьому генерується значення EOF, що вказує на завершення введення текстової інформації. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації. Число може бути складовим певного слова, або записуватися окремо. Якщо в тексті зустрічається комбінація цифрових символів, що розділені крапкою, то програма повинна сприйняти таку цифрову послідовність як два окремих цілих числа. Наприклад, в наступному

	тексті: addr897.74 mode-102 23.1934 55-A unionCP, програма повинна розпізнати як цілі числа наступні цифрові комбінації: 897 74 -102 23 1934 55 Програма підраховує кількість виявлених цілих чисел і **записує в текстовий файл**. Для наведеного прикладу програма повинна записати в текстовий файл число 6.
8.	Написати програму, яка знаходить найбільше серед цілих чисел, що зустрічаються в тексті. Користувач **вводить текст з клавіатури**. Введення завершується натисканням комбінації клавіш Cntrl+Z, при цьому генерується значення EOF, що вказує на завершення введення текстової інформації. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації. Число може бути складовим певного слова, або записуватися окремо. Якщо в тексті зустрічається комбінація цифрових символів, що розділені крапкою, то програма повинна сприйняти таку цифрову послідовність як два окремих цілих числа. Наприклад, в наступному тексті: addr897.74 mode-102 23.1934 55-A unionCP, програма повинна розпізнати як цілі числа наступні цифрові комбінації: 897 74 -102 23 1934 55 Програма знаходить найбільше серед розпізнаних цілих чисел і **записує значення в текстовий файл.** Для наведеного прикладу, програма повинна записати число 1934.
9.	Написати програму, яка знаходить **в текстовому файлі** (файл з

	розширенням *.txt) найдовше слово. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації, однак набір лише символів пунктуації, які обмежені пробільними символами, словом не вважаються. Для створення текстового файлу користуватися редактором Блокнот. Текст може складатися із символів латинського алфавіту, цифрових символів, знаків пунктуації, пробільних символів. Знайдене найдовше слово **записується в текстовий файл.**
10.	Написати програму, яка підраховує скільки разів кожна літера латинського алфавіту зустрічається у **текстовому файлі** (файл з розширенням txt) і **виводить на екран** 8 букв, що найчастіше зустрічаються у файлі, а також виводить скільки разів кожна з них зустрічається в файлі. Для створення текстового файлу користуватися редактором Блокнот.
11.	Написати програму, яка розраховує скільки разів **в текстовому файлі** зустрічаються літери латинського алфавіту a e i o u y. Текстовий файл складається із букв латинського алфавіту, цифрових символів, знаків пунктуації та пробільних символів. Результати програма **виводить на екран**. Для створення текстового файлу користуватися редактором Блокнот.
12.	Написати програму, яка знаходить скільки разів відповідний цифровий символ зустрічається **в текстовому файлі**. Текстовий файл складається із букв латинського алфавіту, цифрових символів, знаків пунктуації та пробільних символів. Результати програма **виводить на екран**. Для створення текстового файлу користуватися редактором Блокнот.

13.	Написати програму, яка підраховує скільки разів кожна літера латинського алфавіту зустрічається у **текстовому файлі** (файл з розширенням txt) і **виводить на екран** 5 букв, що найрідше зустрічаються у файлі, а також виводить скільки разів кожна з них зустрічається в файлі. Для створення текстового файлу користуватися редактором Блокнот.
14.	Написати програму, яка знаходить чотири цифрових символи, що найчастіше зустрічається **в текстовому файлі**. Текстовий файл складається із букв латинського алфавіту, цифрових символів, знаків пунктуації та пробільних символів. Програма **виводить на екран** знайдені цифрові символи, а також скільки разів кожен з них зустрічається в файлі. Для створення текстового файлу користуватися редактором Блокнот.
15.	Написати програму, яка підраховує кількість слів **в текстовому файлі.** Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації, однак набір лише символів пунктуації, які обмежені пробільними символами, словом не вважаються. Наприклад, якщо у файлі зберігається наступний текст: Gen //// B2 190- hello [Cntrl+Z], то програма повинна розпізнати 4 слова. Результат підрахунку кількості слів **виводиться на екран.** Для створення текстового файлу користуватися редактором Блокнот.
16.	Написати програму, яка підраховує кількість знаків пунктуації, що

	зустрічаються в тексті, що зберігається в **текстовому файлі**. Знаками пунктуації вважати наступні символи: знак оклику, кома, крапка, тире, знак питання, двокрапка, крапка з комою. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації. Програма **виводить на екран** знайдену кількість знаків пунктуації. Для створення текстового файлу користуватися редактором Блокнот.
17.	Написати програму, яка підраховує кількість цілих чисел, що зустрічаються в тексті, що зберігається в **текстовому файлі**. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації. Число може бути складовим певного слова, або записуватися окремо. Якщо в тексті зустрічається комбінація цифрових символів, що розділені крапкою, то програма повинна сприйняти таку цифрову послідовність як два окремих цілих числа. Наприклад, в наступному тексті: qw78.78 level_10256 567.567 hello, програма повинна розпізнати як цілі числа наступні цифрові комбінації: 78 78 10256 567 567 Програма підраховує кількість виявлених цілих чисел і **виводить на екран**. Для наведеного прикладу програма повинна записати в текстовий файл число 5. Для створення текстового файлу користуватися редактором Блокнот.
18.	Написати програму, яка знаходить найбільше серед цілих чисел, що

	зустрічаються в тексті що зберігається в **текстовому файлі**. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації. Число може бути складовим певного слова, або записуватися окремо. Якщо в тексті зустрічається комбінація цифрових символів, що розділені крапкою, то програма повинна сприйняти таку цифрову послідовність як два окремих цілих числа. Наприклад, в наступному тексті: qw78.78 level_10256 567.567 hello, програма повинна розпізнати як цілі числа наступні цифрові комбінації 78 78 10256 567 567 Програма знаходить найбільше серед розпізнаних цілих чисел і **виводить на екран.** Для наведеного прикладу, програма повинна записати число 10256.
19.	Написати програму, яка підраховує кількість слів **введених із клавіатури.** Програма виводить запит на введення тексту із клавіатури. Користувач вводить текст. Введення завершується натисканням комбінації клавіш Cntrl+Z, при цьому генерується значення EOF, що вказує на завершення введення текстової інформації. Текст, що вводиться з клавіатури складається із символів латинського алфавіту, цифрових символів, знаків пунктуації, пробільних символів. Підрахувати скільки слів було введено користувачем, при цьому окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації, однак набір лише символів пунктуації, які обмежені пробільними символами, словом не вважаються. Наприклад, якщо користувач ввів з клавіатури наступну послідовність: sky black

	999 .f.!4! [Cntrl+Z], то програма повинна розпізнати 3 слова. Результат підрахунку кількості слів **записується в текстовий файл.**
20.	Написати програму, яка знаходить **в текстовому файлі** (файл з розширенням *.txt) найдовше слово. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації, однак набір лише символів пунктуації, які обмежені пробільними символами, словом не вважаються. Для створення текстового файлу користуватися редактором Блокнот. Знайдене найдовше слово **виводиться на екран.**
21.	Написати програму, яка підраховує кількість символів, з яких складається найдовше слово в тексті, що **вводиться із клавіатури**. Користувач вводить текст. Введення завершується натисканням комбінації клавіш Cntrl+Z, при цьому генерується значення EOF, що вказує на завершення введення текстової інформації. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації, однак набір лише символів пунктуації, які обмежені пробільними символами, словом не вважаються. Програма **записує в текстовий файл** кількість символів, з яких складається найдовше слово.
22.	Написати програму, яка підраховує скільки разів кожна літера латинського алфавіту зустрічається у **текстовому файлі** (файл з розширенням txt) і **виводить на екран** 8 букв, що найчастіше зустрічаються у файлі, а також виводить скільки разів кожна з них

	зустрічається в файлі. Для створення текстового файлу користуватися редактором Блокнот.
23.	Написати програму, яка підраховує кількість слів **в текстовому файлі.** Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації, однак набір лише символів пунктуації, які обмежені пробільними символами, словом не вважаються. Наприклад, якщо у файлі зберігається наступний текст: Ars „.!? 5Fg2 583! bye [Cntrl+Z], то програма повинна розпізнати 4 слова. Результат підрахунку кількості слів **виводиться на екран.** Для створення текстового файлу користуватися редактором Блокнот.
24.	Написати програму, яка знаходить чотири цифрових символи, що найчастіше зустрічається **в текстовому файлі**. Текстовий файл складається із букв латинського алфавіту, цифрових символів, знаків пунктуації та пробільних символів. Програма **виводить на екран** знайдені цифрові символи, а також скільки разів кожен з них зустрічається в файлі. Для створення текстового файлу користуватися редактором Блокнот.
25.	Написати програму, яка розраховує скільки разів **в текстовому файлі** зустрічаються літери латинського алфавіту z y x w v . Текстовий файл складається із букв латинського алфавіту, цифрових символів, знаків пунктуації та пробільних символів. Результати програма **виводить на екран**. Для створення текстового файлу користуватися редактором Блокнот.

26.	Написати програму, яка підраховує кількість цифр, що зустрічаються в тексті. Користувач вводить **текст з клавіатури**. Введення завершується натисканням комбінації клавіш Cntrl+Z, при цьому генерується значення EOF, що вказує на завершення введення текстової інформації. Окремим словом вважається набір символів, що не містить пробільних символів. Таким чином, слово може бути сформоване як із букв латинського алфавіту і/або цифр, а також включати знаки пунктуації. Програма **записує в текстовий файл** знайдену кількість цифрових символів.