МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

І.М. Гаркуша

ПРОГРАМУВАННЯ

Методичні рекомендації до виконання лабораторних робіт
для здобувачів ступеня бакалавра освітньо-професійних програм
«Інформаційні системи та технології»
спеціальності 126 Інформаційні системи та технології
та «Комп'ютерна інженерія»
спеціальності 123 Комп'ютерна інженерія

Дніпро
НТУ «ДП»
2025

Гаркуша І. М.

Програмування [Електронний ресурс] : методичні рекомендації до виконання лабораторних робіт для здобувачів ступеня бакалавра освітньо-професійних програм «Інформаційні системи та технології» спеціальності 126 Інформаційні системи та технології та «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія / І. М. Гаркуша ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2025. – 66 с.

Автор
І. М. Гаркуша, канд. техн. наук, доц.

Затверджено науково-методичними комісіями спеціальностей 126 Інформаційні системи та технології та 123 Комп'ютерна інженерія (протокол № 1 від 14.04.2025) за поданням кафедри інформаційних технологій та комп'ютерної інженерії (протокол № 13 від 11.04.2025).

Надані теоретичні відомості та практичні завдання, а також рекомендації щодо їх виконання. Поданий перелік рекомендованих джерел.

Орієнтовано на активізацію навчальної діяльності здобувачів ступеня бакалавра спеціальностей «Інформаційні системи та технології» та «Комп'ютерна інженерія», закріплення практичних навичок у засвоєнні дисципліни «Програмування».

Відповідальний за випуск завідувач кафедри інформаційних технологій та комп'ютерної інженерії В. В. Гнатушенко, д-р техн. наук, проф.

ЗМІСТ

ВСТУП 4

Робота №1. Опис алгоритмів за допомогою схем 5

Робота №2. Програмування обчислень математичних виразів мовою С 17

Робота №3. Програмування алгоритмів розгалуження мовою С 21

Робота №4. Програмування циклічних алгоритмів мовою С 28

Робота №5. Робота з динамічними векторами та матрицями 39

Робота №6. Робота з файлами 43

Самостійна робота 48

Критерії оцінювання виконання робіт 50

Перелік рекомендованих джерел 51

Додаток А. Вимоги до оформлення звітності 52

Додаток Б. Деякі функції стандартної бібліотеки мови С 56

ВСТУП

Мова програмування С має вплив на розвиток програмного забезпечення (ПЗ) в ІТ-галузі. На відміну від існуючих популярних мов розробки ПЗ, вона все також є легкою в засвоєнні та надає базу для розуміння інших мов програмування. Мова стандартизована. Станом на 2025 рік актуальною є редакція стандарту мови – С23. Як і багато років тому, починаючи з 1970-х років, мова С є основною мовою системного програмування. Переважна більшість сучасних операційних систем (ОС), їх ядра, компоненти, елементи інтерфейсів користувача, різноманітні допоміжні бібліотеки, написані мовою С. Все також цією мовою програмують різноманітні системи керування, апаратні контролери, створюють ігри, програмують мультимедіа системи та тривимірну графіку. За світовим рейтингом мов розробки, мова С займає топові позиції (входить в десятку рейтингових мов програмування у світі). У переважній більшості найтитульованіших університетів світу ця мова є обов'язковою для вивчання здобувачами освіти, які опановують професії ІТ-галузі.

Методичні рекомендації до виконання робіт з практикуму програмування мовою С є частиною курсу «Програмування», який викладається в Національному технічному університеті «Дніпровська політехніка» здобувачам спеціальностей 123 Комп'ютерна інженерія та 126 Інформаційні системи та технології в першому семестрі першого курсу навчання.

Методичні рекомендації пропонують виконання шести основних робіт та одного додаткового самостійного завдання. Перша робота присвячена питанню побудови та розумінню схем алгоритмів і має за основну мету – повторення доуніверситетських базових знань з алгоритмізації, а також закріплення практичних навичок побудови схем алгоритмів згідно Національного стандарту України ДСТУ ISO 5807:2016 автоматизованими засобами. Інші практичні роботи присвячені набуттю навичок створення комп'ютерних консольних програм мовою програмування С.

В якості середовища розробки пропонується *Eclipse IDE* (Integrated Development Environment – інтегроване середовище розробки). Це середовище є крос-платформним (існує під різні ОС: MS Windows, GNU/Linux-сумісні, Unix-подібні – у тому числі Apple macOS). Eclipse IDE використовує відносно невеликі об'єми дискової та оперативної пам'яті та легке у засвоєнні повноцінне IDE. Середовище займає високі рейтинги по використанню у всесвітніх компаніях з розробки ПЗ та має періодичне оновлення й підтримку. Окрім того, оскільки здобувачі курсу в останні роки мають власні апаратні засоби (ноутбуки, настільні комп'ютери) та можуть опановувати навчальні матеріали дистанційно, працюючи під різними ОС, то середовище Eclipse IDE фактично виступає універсальним інструментом.

В якості основного компілятору обраний GNU GCC (для платформи MS Windows – зі складу проєкту MinGW-w64) або схожий для певних ОС, наприклад, компілятор Clang.

Робота №1.
Опис алгоритмів за допомогою схем

Мета роботи: повторення доуніверситетських базових знань з алгоритмізації та закріплення практичних навичок побудови схем алгоритмів згідно Національного стандарту України ДСТУ ISO 5807:2016.

Теоретична частина

Найпростішим різновидом алгоритмів є алгоритми лінійної структури, в яких операції виконуються послідовно один за одним. Наприклад:

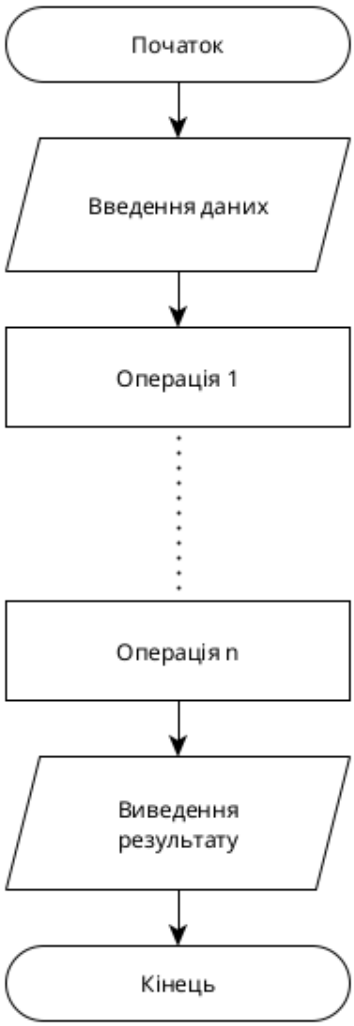
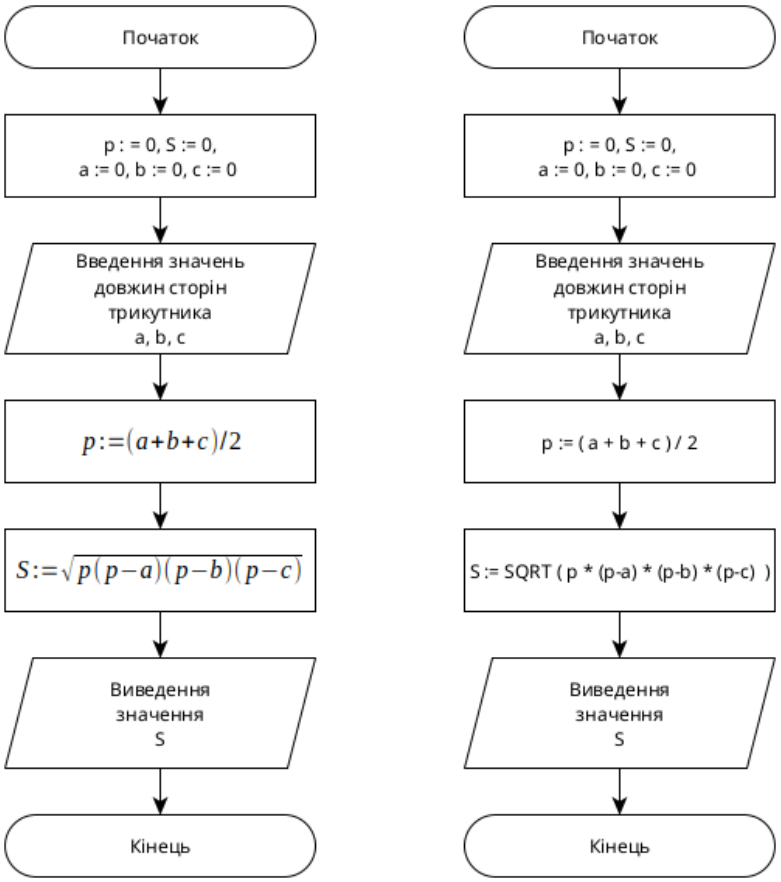


Рис. 1.1. Приклад загальної схеми алгоритму лінійної структури, представлений графічними символами з Національного стандарту України ДСТУ ISO 5807:2016

На рис. 1.2 представлені приклади схем алгоритму обчислення площі трикутника за умови, коли відомі довжини трьох його сторін.

Іншим різновидом алгоритмів, які широко використовуються, є алгоритми, що передбачають кілька різних шляхів вирішення певної частини поставленого завдання – алгоритми з розгалуженням.



а) б)

Рис. 1.2. Схеми алгоритму обчислення площі трикутника за трьома сторонами в подані із залученням математичного символу обчислення кореня квадратного (а) та варіант з використанням програмної функції обчислення кореня квадратного (б) із загальною назвою SQRT

Згідно ДСТУ ISO 5807:2016 існує кілька форм представлення алгоритмів з розгалуженням (рис. 1.3). Слід зазначити, що символи вибору рішення можуть бути вкладеними. Тобто відразу після вибору рішення (відповідь «Так» або «Ні») може відразу йти наступний символ вибору рішення. Вкладеність залежить від складності алгоритму. Однак не слід використовувати дуже глибоку вкладеність. Якщо це відбувається, то слід обрати або останню форму подання, яка зображена на рис. 1.3, або спробувати перебудувати умову(и), або перебудувати алгоритм.

Наприклад, розробимо схему алгоритму знаходження коренів квадратного рівняння $ax^2+bx+c=0$ за умови, коли $a\neq 0$ – рис. 1.4. Перший символ вибору рішення є обов'язковим, оскільки алгоритм передбачає в подальшому операції ділення на величину $2a$. Другий символ вибору рішення перевіряє знайдений дискримінант – його значення повинно бути більше або дорівнювати нулю.

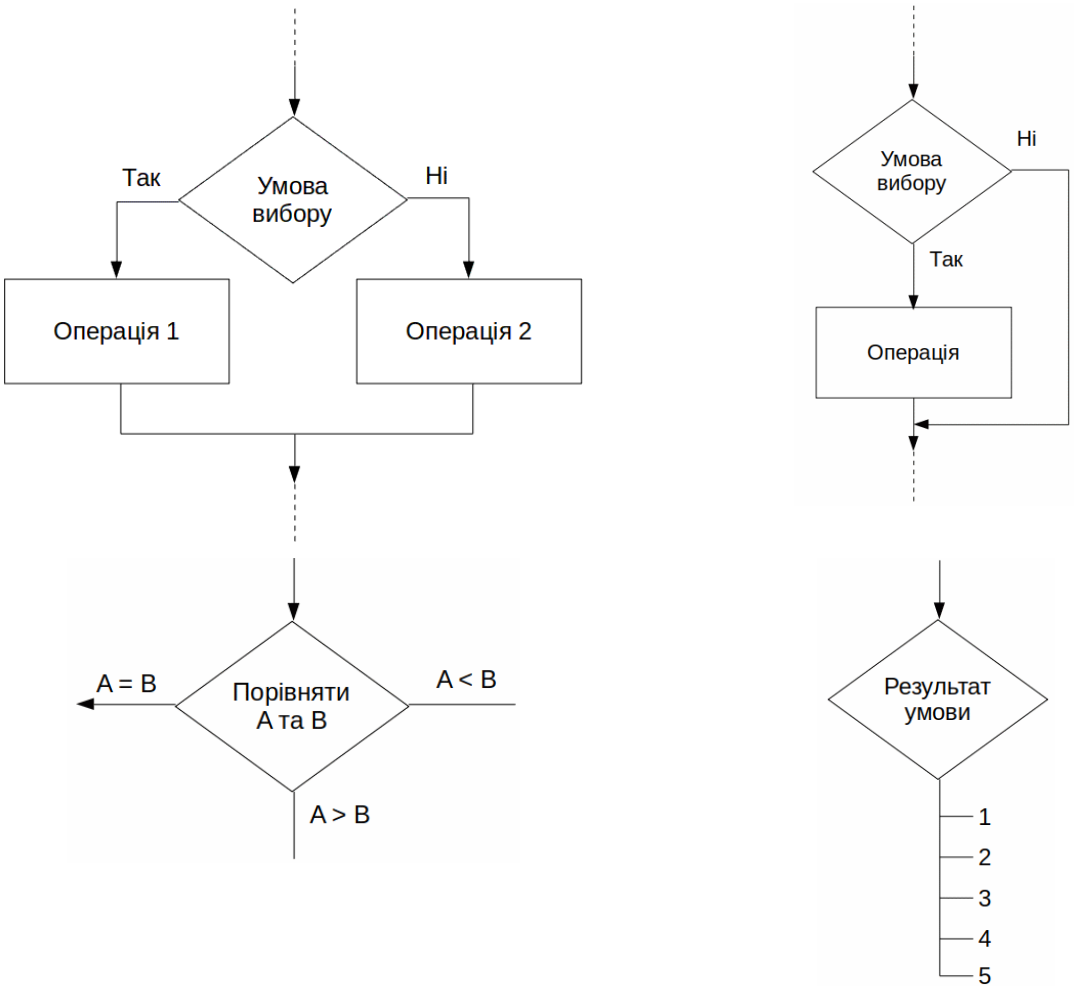


Рис. 1.3. Приклади зображення символу вибору рішення в алгоритмах розгалуження

Третім різновидом алгоритмів є алгоритми циклічної структури, які передбачають повторення певних операцій визначену кількість разів. Проходження програмою операторів в циклі один раз, називають ітерацією циклу. Таким чином, цикл являє собою виконання безлічі ітерацій по заданій умові.

Слід зазначити, що цикли можуть бути «нескінченними» – тобто якщо в якості умови виконання циклу буде завжди величина, яка призводить до виконання наступної ітерації циклу. Однак, навіть в таких циклах повинні бути умови завершення циклу – виходу з нього. В даному випадку символи вибору рішення, як правило, розміщують всередині циклу.

Різні варіанти представлення схем алгоритмів з циклічною структурою наведені на рис. 1.5.

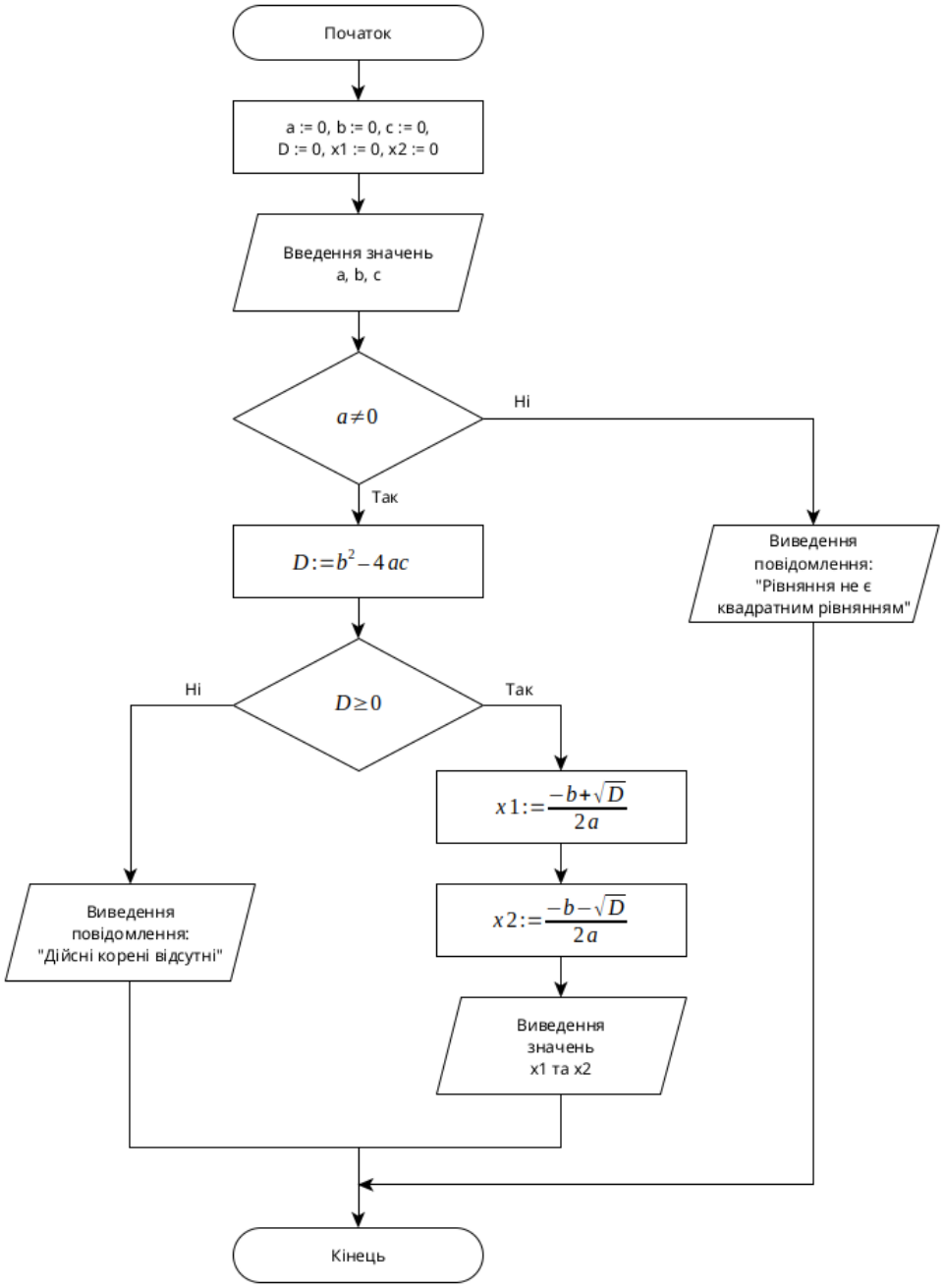


Рис. 1.4. Схема алгоритму знаходження коренів квадратного рівняння $ax^2+bx+c=0$

Варто звернути увагу на такі основні моменти.

1. При виконанні циклу з післяумовою (рис. 1.5, а) оператори тіла циклу будуть виконані хоча б один раз.

2. При реалізації циклу з передумовою (рис. 1.5, б), якщо умова не буде виконана, то весь цикл не буде виконаний.

3. Схема алгоритму на рис. 1.5, в) дозволяє організувати ітераційний цикл з лічильником, при цьому можливий варіант використання такого уявлення циклу і для двох зазначених вище випадків.

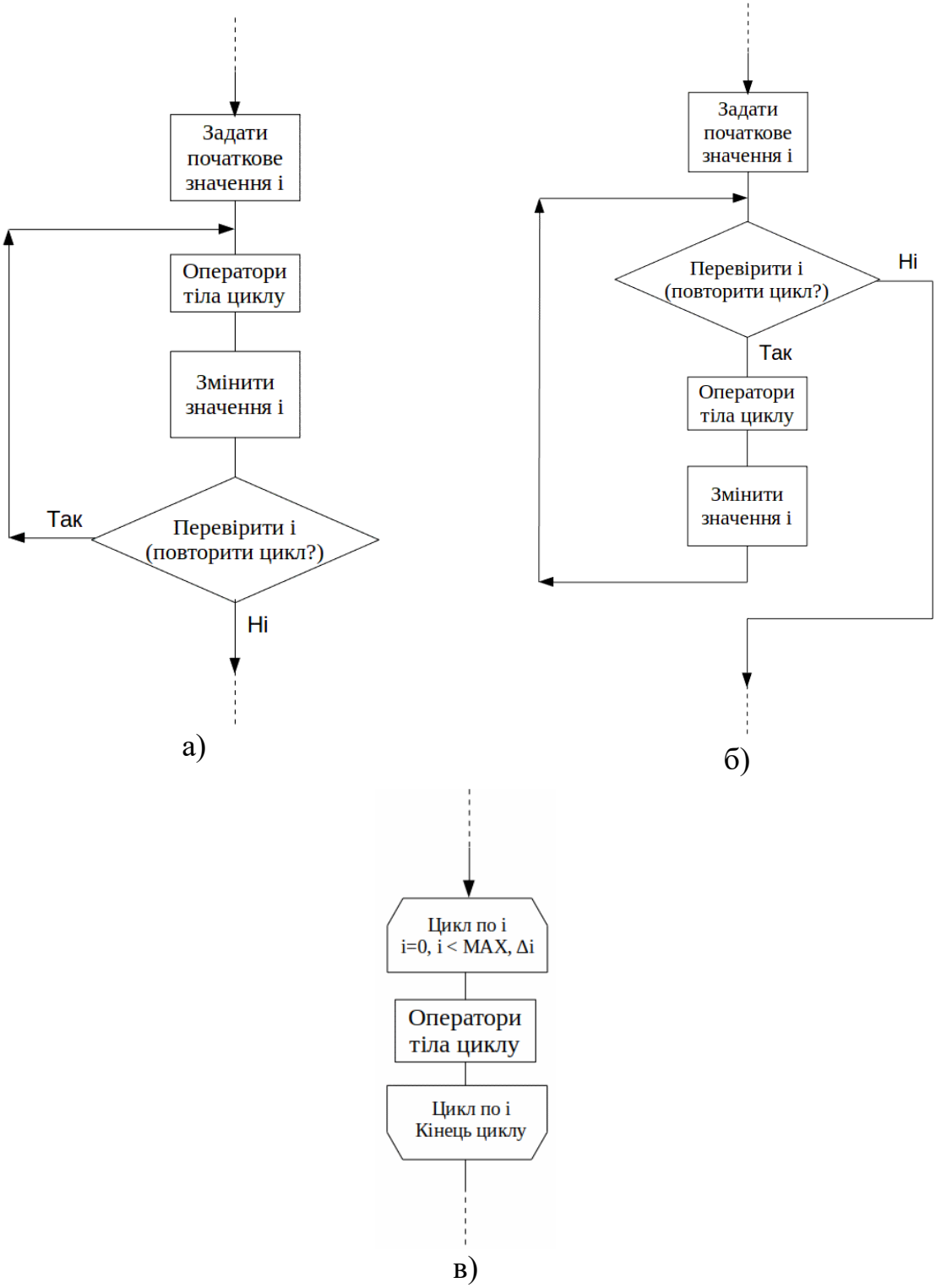


Рис. 1.5. Різновиди представлення циклічних схем алгоритмів:
а) цикл з післяумовою; б) цикл з передумовою;
в) ітераційний цикл з лічильником

4. Цикли можуть бути вкладеними. Тобто в якості операторів тіла циклу може бути інша циклічна структура. **Слід зазначити, що підвищуючи вкладеність циклів при обробці великих обсягів даних, програміст значно збільшує тривалість за часом обробки таких даних!**

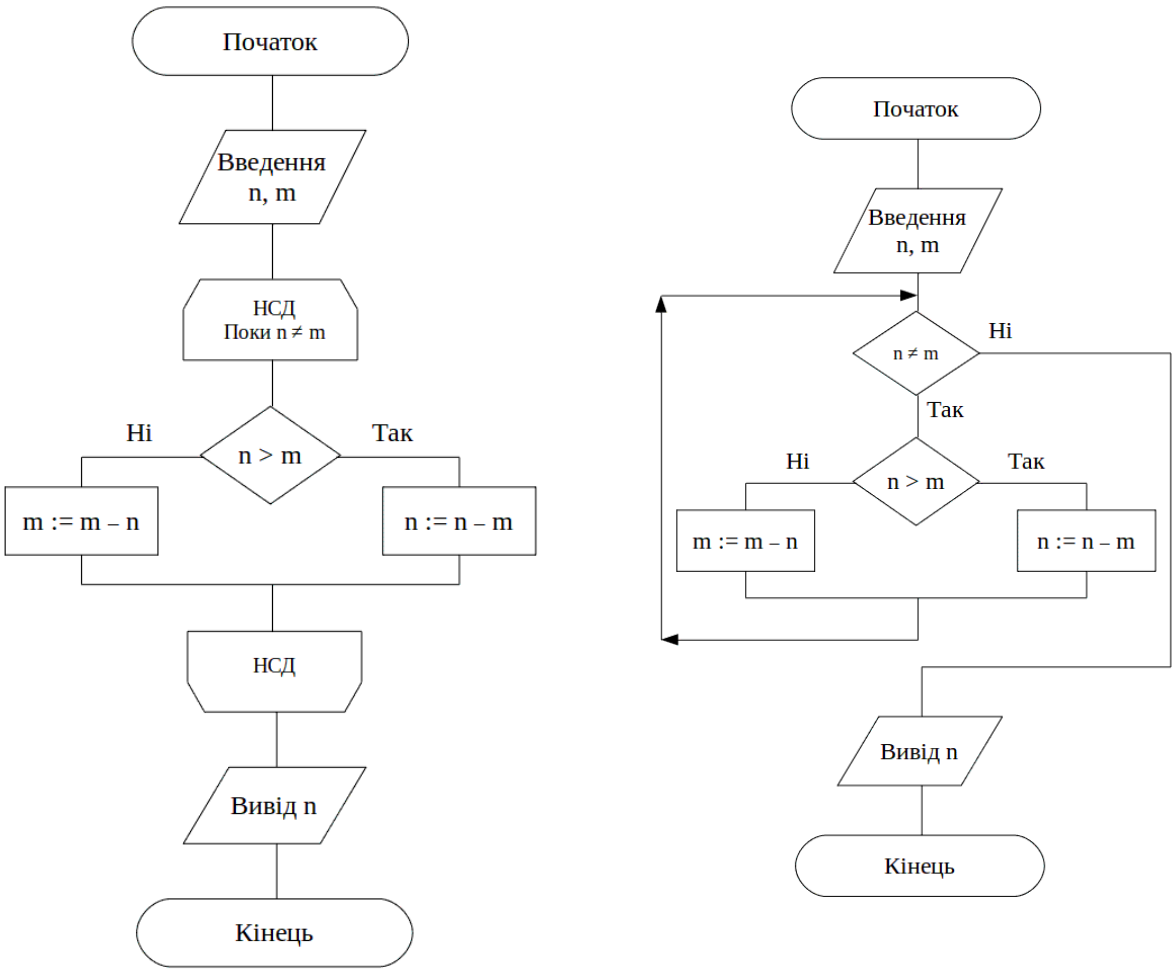


Рис. 1.6. Приклади різних зображень схеми алгоритму знаходження найбільшого спільного дільника (НСД)

Як приклад реалізації циклічного алгоритму, розглянемо алгоритм знаходження найбільшого спільного дільника (НСД) двох позитивних цілих чисел (рис. 1.6), відомий як алгоритм Евкліда. Алгоритм може бути описаний у словесній формі. Наприклад, у випадку зі знаходженням НСД маємо:

Крок 1. Дано два цілих позитивних числа n та m.

Крок 2. Якщо $n \neq m$, то переходимо до пошуку на крок 3. Інакше якщо $n = m$, то n буде НСД і переходимо до кроку 6.

Крок 3. Порівнюємо n з m та обираємо більший з них.

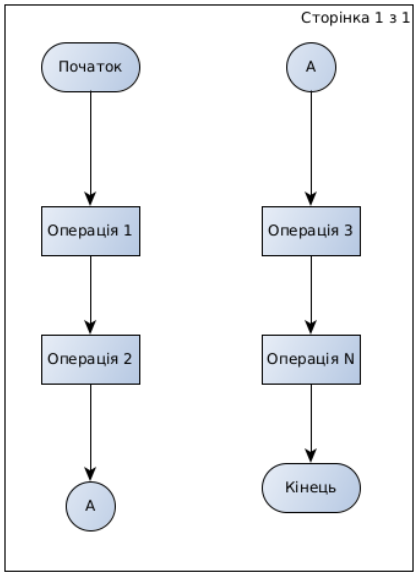
Крок 4. Більше значення з пари n, m замінюємо різницею більшого та меншого.

Крок 5. Переходимо до кроку 2.

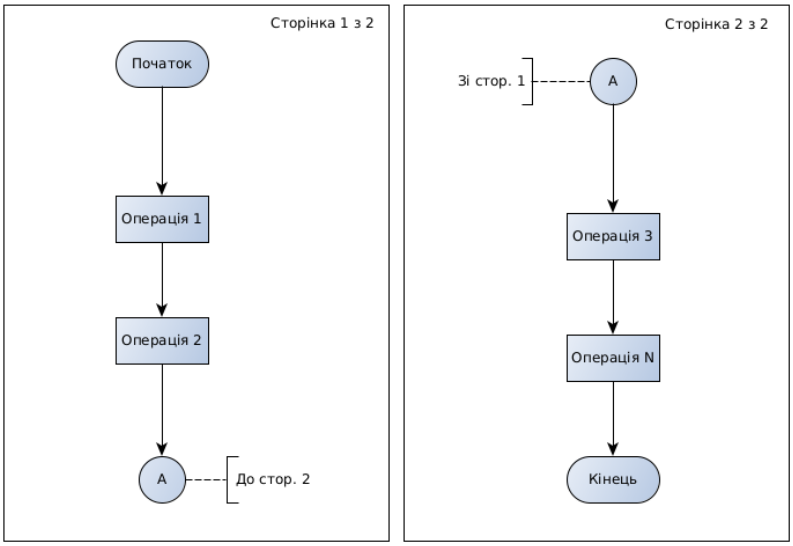
Крок 6. Виведення *n* – знайдений НСД.
Крок 7. Кінець алгоритму.

Зверніть увагу, що стрілки-покажчики у схемах алгоритмів додають при необхідності або для підвищення зручності читання напрямку потоку обробки. За ДСТУ ISO 5807:2016 стандартний напрямок потоку обробки в алгоритмах – зліва направо та зверху вниз. Якщо потік має напрямок, який відмінний від стандартного, то стрілки повинні вказувати цей напрямок.

Якщо схема алгоритму завелика та потрібно рознести її по ширині аркуша або перенести на іншій аркуш, то використовують графічні символи та відображення, які показані на рис. 1.7.



а)



б)

Рис. 1.7. Правила перенесення частини схеми алгоритму на одній сторінці аркуша (а) та між декількома сторінками (б)

Як бачимо у випадку декількох аркушів є доречним коментування позначки перенесення із вказівкою номера сторінки, на якій йде продовження схеми алгоритму.

Завдання

1. Побудувати схему алгоритму лінійної структури для обчислення виразу згідно варіанту завдання з таблиці 1.1, ***вважаючи за замовчуванням, що знаменники у виразах не будуть дорівнювати нулю, а вхідні значення змінних будуть більшими за нуль***.

Таблиця 1.1

Варіанти завдань для побудови
схеми алгоритму лінійної структури

№ варіанту	*Вираз*
1	$S1 = xyz + 2{,}54\dfrac{32x - 17x\,2z + 12y}{y(z - 3{,}27y)}$
2	$S1 = \dfrac{21x - 5y + 37z - x/y}{x + y/z - 25y}$
3	$S1 = \dfrac{x - 2{,}24\,yz/x - 5}{x - y + 1{,}6z} + 12x$
4	$S1 = z/x - 1{,}25\dfrac{x + 37y - 7{,}2z/x}{7z}$
5	$S1 = \dfrac{x + z - 3z/x}{7z} + \dfrac{2x + 12y - 3{,}5z/x}{3{,}5z}$
6	$S1 = xyz - \dfrac{32x - 68xyz + 12y/x}{2{,}5y(2z - y)}$
7	$S1 = \dfrac{1{,}2x + 3{,}4y - 12z - y/z}{2xy/z + 15y}$
8	$S1 = \dfrac{25x - 36y^2/(2z) - 0{,}25}{2x + 3y - 4z} - 2{,}3x$
9	$S1 = 2x/y^2 + 0{,}75\dfrac{x^2 - 12y + 3{,}4y/z}{5x}$
10	$S1 = \dfrac{2x - 5{,}7z + z/x}{3x} - \dfrac{3x + 5y - 2{,}5y/z}{10y^2}$
11	$S1 = \dfrac{2yz}{5x} - \dfrac{32xz}{2{,}5y} + \dfrac{5{,}2xy}{2{,}1z}$
12	$S1 = \cos(2x) - \sin(3{,}4y) + \cos(3x) - \sin(5{,}4z)$

13	$S1=\frac{\cos(2x)\sin(3{,}4y)}{\cos(3y)\sin(5{,}4z)}$
14	$S1=2x^2-3{,}4y^3+5{,}4z^5$
15	$S1=\frac{2x^2}{3{,}5y^3}-\frac{3{,}4y^3}{5{,}4z^5}$
16	$S1=e^{2x}-\cos(3{,}4y^3)+5{,}4\sin(z^2)$
17	$S1=\frac{\sin\lvert 1-\ln(x)\rvert+\cos(2{,}5y)}{\sin^2(3{,}4z)}$ довідникова інформація: https://en.wikipedia.org/wiki/Logarithm
18	$S1=\frac{\cos(2x)}{1-\sin(2y)}+\frac{\sin(3y)}{1+\cos(4z)}$
19	$S1=\frac{\cos(2x)-\sin(3{,}4y)+\cos(3z)}{2{,}5x^2+0{,}85z^3}+\sin(5{,}4z)$
20	$S1=\frac{2x^2-3{,}4y^3}{2y^2-3{,}4z^3}+5{,}4z^3$

2. Побудувати схему алгоритму з розгалуженням для обчислення виразу згідно варіанту завдання з таблиці 1.2, ***що враховує випадки рівності нулю знаменника виразів або ж негативних значень під квадратним коренем***.

Таблиця 1.2

Варіанти завдань для побудови
схеми алгоритму з розгалуженням

№ варіанту	*Вираз*
1	$S2=xyz+2{,}54\frac{32x-17x2z+12y}{y(z-3{,}27y)}$
2	$S2=\frac{21x-5y+37z-x/y}{x+y/z-25y}$
3	$S2=\frac{x-2{,}24yz/x-5}{x-y+1{,}6z}+12x$
4	$S2=z/x-1{,}25\frac{x+37y-7{,}2z/x}{7z}$
5	$S2=\frac{x+z-3z/x}{7z}+\frac{2x+12y-3{,}5z/x}{3{,}5z}$
6	$S2=xyz-\frac{32x-68xyz+12y/x}{2{,}5y(2z-y)}$
7	$S2=\frac{1{,}2x+3{,}4y-12z-y/z}{2xy/z+15y}$

8	$S2=\frac{25x-36y^2/(2z)-0,25}{2x+3y-4z}-2,3x$
9	$S2=2x/y^2+0,75\frac{x^2-12y+3,4y/z}{5x}$
10	$S2=\frac{2x-5,7z+z/x}{3x}-\frac{3x+5y-2,5y/z}{10y^2}$
11	$S2=\frac{2yz}{5x}-\frac{32xz}{2,5y}+\frac{5,2xy}{2,1z}$
12	$S2=\frac{\cos(2x)\sin(3,4y)}{\cos(3y)\sin(5,4z)}$
13	$S2=\frac{2x^2}{3,5y^3}-\frac{3,4y^3}{5,4z^5}$
14	$S2=\frac{\cos(2x)}{1-\sin(2y)}+\frac{\sin(3y)}{1+\cos(4z)}$
15	$S2=\frac{2x^2-3,4y^3}{2y^2-3,4z^3}+5,4z^3$
16	$S2=\frac{2x^2-3,4y^3}{\sqrt{2y^2-3,4z^3}}+\sqrt{5,4z^3}$
17	$S2=\frac{\cos(2x)\sin(3,4y)}{\sqrt{\cos(2y)}+\sqrt{\sin(5,4z)}}$
18	$S2=\frac{2x-5,7z+z/x}{\sqrt{3x}}-\frac{3x+5y-2,5y/z}{\sqrt{10y^2}}$
19	$S2=\cos(2x)-\sin(3,4y)+\sqrt{\cos(3x)}-\sqrt{\sin(5,4z)}$
20	$S2=\frac{\cos(2x)-\sin(3,4y)}{\sqrt{\cos(3x)}+\sqrt{\sin(5,4z)}}$

3. Побудувати схему циклічного алгоритму для обчислення виразу згідно варіанту завдання з таблиці 1.3. Відомо, що вхідні значення повинні лежати в діапазоні: $0<a_i<2\pi$. ***Це також потрібно врахувати при побудові схеми алгоритму***.

Таблиця 1.3

Варіанти завдань для побудови схеми циклічного алгоритму

№ варіанту	*Вираз*
1	$S3=\frac{\sum_{i=1}^{5}\sin\left\|1-\ln(a_i)\right\|}{\sum_{i=1}^{11}\sin^2 18a_i^3}$ довідникова інформація: https://en.wikipedia.org/wiki/Logarithm

2	$$S3=\frac{\ln\sum_{i=1}^{11}\left(\left\|\sin a_i\right\|^{\sin a_i}+\left\|\cos a_i\right\|^{\cos a_i}\right)}{3\sum_{i=1}^{11}\sin^2 a_i^{\sin a_i}}$$ довідникова інформація: https://en.wikipedia.org/wiki/Logarithm
3	$$S3=\sum_{i=1}^{11}\frac{\cos\left(2a_i\right)}{1-\sin\left(2a_i\right)}$$
4	$$S3=\sqrt{\frac{\sum_{i=1}^{11}\left(a_i-\bar{a}\right)^2}{10}},\bar{a}=\frac{1}{11}\sum_{i=1}^{11}a_i$$
5	$$S3=\frac{\sin^2\sum_{i=1}^{11}\ln\left\|1-\cos^2 a_i\right\|}{\sum_{i=1}^{11}\cos\left\|1-\sin a_i\right\|}$$ довідникова інформація: https://en.wikipedia.org/wiki/Logarithm
6	$$S3=\sum_{i=1}^{11}\left(\cos\left(2a_i\right)\sin\left(3{,}4a_i\right)\right)$$
7	$$S3=\frac{\sum_{i=1}^{11}\sin\left(1-3\sin^3 a_i\right)}{3\sum_{i=1}^{11}\cos^2 a_i^5}$$
8	$$S3=\sum_{i=1}^{11}\frac{1-2\cos^2 a_i}{\left\|\sin a_i\right\|*\left\|\cos a_i\right\|}$$
9	$$S3=\sum_{i=1}^{11}\left(1+\cos a_i+\cos\frac{a_i}{2}\right)$$
10	$$S3=\sum_{i=1}^{11}\left(1+\sin\frac{a_i^2}{3}+\sin\frac{a_i^3}{4}\right)$$
11	$$S3=\frac{\sum_{i=1}^{4}\sin\left(1-2\cos^2 a_i\right)}{\sum_{i=5}^{11}\sin\frac{a_i^2}{3}}$$
12	$$S3=\frac{lg\left\|\sum_{i=1}^{7}\sin^3 a_i\right\|}{5{,}7\sum_{i=1}^{11}\cos^2 a_i}$$ довідникова інформація: https://en.wikipedia.org/wiki/Logarithm

13	$S3=\sum_{i=1}^{11}\frac{\sin^3 a_i}{1+\cos^2\frac{a_i^3}{2,5}}$
14	$S3=\sum_{i=1}^{11}\frac{1}{\cos^3 a_i}-\sum_{i=1}^{11}\frac{1}{\sin^3 a_i}+\sum_{i=1}^{11} lg\left(a_i^2\right)$ довідникова інформація: https://en.wikipedia.org/wiki/Logarithm
15	$S3=\frac{\sum_{i=1}^{11}\left(12a_i^3+\sin^3 a_i-tg^5 a_i\right)}{\sum_{i=1}^{11}\left(2,7a_i+a_i^{1/3}\right)}$
16	$S3=\sum_{i=1}^{11}\frac{lg\,a_i-\sin a_i+tg\frac{1}{a_i}}{3,4+\left(a_i+1\right)^{1/4}}$ довідникова інформація: https://en.wikipedia.org/wiki/Logarithm
17	$S3=\frac{\sum_{i=1}^{11}\sin^3 a_i}{\sum_{i=1}^{11}\left(1+\cos^2\frac{a_i^3}{2,5}\right)}$
18	$S3=\sum_{i=1}^{11}\frac{lg\left\lvert\sin^3 a_i\right\rvert}{5,7\cos^2 a_i}$ довідникова інформація: https://en.wikipedia.org/wiki/Logarithm
19	$S3=\sum_{i=1}^{11}\frac{\sin\left(1-2\cos^2 a_i\right)}{\sin\frac{a_i^2}{3}}$
20	$S3=\sum_{i=1}^{4}\left(1+\cos a_i\right)+\sum_{i=5}^{11}\left(\cos\frac{a_i}{2}\right)$

У звіт з виконаної роботи включити три розроблених схеми алгоритмів обчислення виразів, згідно ***виданих викладачем варіантів завдань***.

Оформлення схем алгоритмів рекомендується виконувати в спеціалізованих середовищах побудови діаграм/схем (наприклад, yEd, Dia, MS Visio або подібних), у спеціалізованих Web-середовищах (наприклад, https://app.diagrams.net/ (draw.io)) або в іншому ПЗ, яке надає такі можливості. Правила оформлення звіту подані в додатку А.

Робота №2.
Програмування обчислень математичних виразів мовою С

Мета роботи: навчитися розробляти прості консольні програми мовою програмування С. Закріпити знання про використання директиви препроцесора *include*, використання математичних функцій та функції форматованого консольного виведення.

Теоретично-практична частина

Розглянемо побудову алгоритму лінійної структури для обчислення математичного виразу:

$$S1 = \frac{x^3 + 3{,}4\, y^5 - x/z}{1024\, x - 100\, y} \qquad (2.1)$$

Відомо, що за замовчуванням вхідні значення для *x*, *y* та *z* будуть такими, при яких знаменники в (2.1) не будуть дорівнювати нулю.

Алгоритм лінійної структури не містить ніяких частин з розгалуженням або циклічним виконанням.

Нехай вхідні дані для змінних *x*, *y*, *z* будуть такі, що дадуть обчислити математичний вираз за лінійним алгоритмом і ніколи не призведуть до аварійного завершення програми. Наприклад, нехай маємо такі значення: $x = 2.5$, $y = 3.1$, $z = 0.41$.

Тоді алгоритм для обчислення виразу (2.1) можна описати схемою, яка зображена на рис. 2.1.

Тобто спочатку за алгоритмом потрібно мати декілька змінних, в які будуть закладені початкові значення. Оскільки маємо вхідні значення з плаваючою комою, то в якості типу даних для змінних можна обрати, наприклад, тип *float* або *double*. В даному прикладі вибір типу не є принциповим. Тому можна обрати, наприклад, тип *double*.

Розрахунок виразу (2.1) можна вести як в одному операторному рядку, так і розбити на частини. Згідно схеми алгоритму (рис. 2.1), результат обчислення чисельника буде містити тимчасова змінна *a*. Результат обчислення знаменника – тимчасова змінна *b*. Оскільки результати обчислень будуть не цілими значеннями, то типи даних для *a* та *b* можна також обрати як *double*.

Оскільки у математичному виразі є операції возведення в ступінь, то можна скористатися викликом відповідної математичної функції `pow()`, заголовок якої знаходиться у файлі *math.h* (додаток Б, таблиця Б.5) із складу стандартної бібліотеки мови С.

Таким чином, згідно схеми алгоритму, маємо фрагмент коду мовою С:

```
. . .
double x = 2.5, y = 3.1, z = 0.41;  // змінні із вхідними даними
double S1 = 0;                      // змінна для збереження результату
double a = 0, b = 0;                // тимчасові змінні

// розраховуємо чисельник
a = pow(x, 3.) + 3.4 * pow(y, 5.) - x / z;
// розраховуємо знаменник
b = 1024 * x - 100 * y;
// розраховуємо результат обчислення математичного виразу
S1 = a / b;
. . .
```

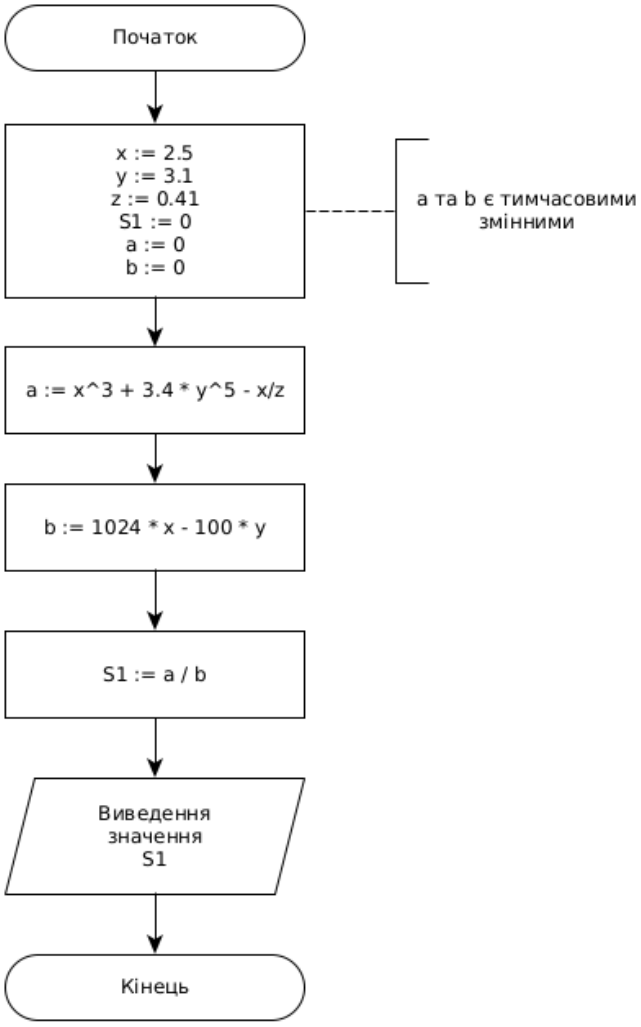


Рис. 2.1. Схема алгоритму лінійної структури для обчислення математичного виразу (2.1)

Виведення результату обчислення роблять функцією форматованого виведення `printf()`, заголовок якої оголошений у файлі *stdio.h* (додаток Б, таблиця Б.1) із складу стандартної бібліотеки мови С.

Приклад виклику функції `printf()`:

```
. . .
// виведення результату обчислення – значення змінної S1
printf("S1 = %lf\n", S1);
```

При формуванні рядка виведення значення на консоль, використовуємо специфікатор формату типа *double*: `%lf`, замість якого функція підставить значення із змінної *S1*. Спецсимвол (символ керування курсором) `'\n'` призведе до переведення курсору на новий рядок у вікні термінала консолі.

Примітка 2.1: для типу *double* специфікатор формату `%lf` представлений в редакції стандарту С99. В новітніх редакціях стандарту можна використовувати специфікатор формату `%f`. Тобто `%lf` використаний для сумісності зі старою редакцією. Для виведення значень типу *long double* використовують специфікатор формату `%Lf`. Специфікатор формату для типу *float*, в сучасних редакціях стандарту такий, як у *double* – наприклад, `%f`. Для функції форматованого введення `scanf()` специфікатори формату для типів *float* та *double* відрізняються.

Всі ці міркування потрібно сформувати в окремий файл коду мовою С (наприклад, *Laba2.c*) з врахуванням підключення потрібних файлів заголовків за допомогою директиви препроцесора *include*, та включити в реалізацію головної функції програми (головної точки входу в програму) – у функцію `main()`. Завершений код програми розрахунку (2.1) представлений на рис. 2.2.

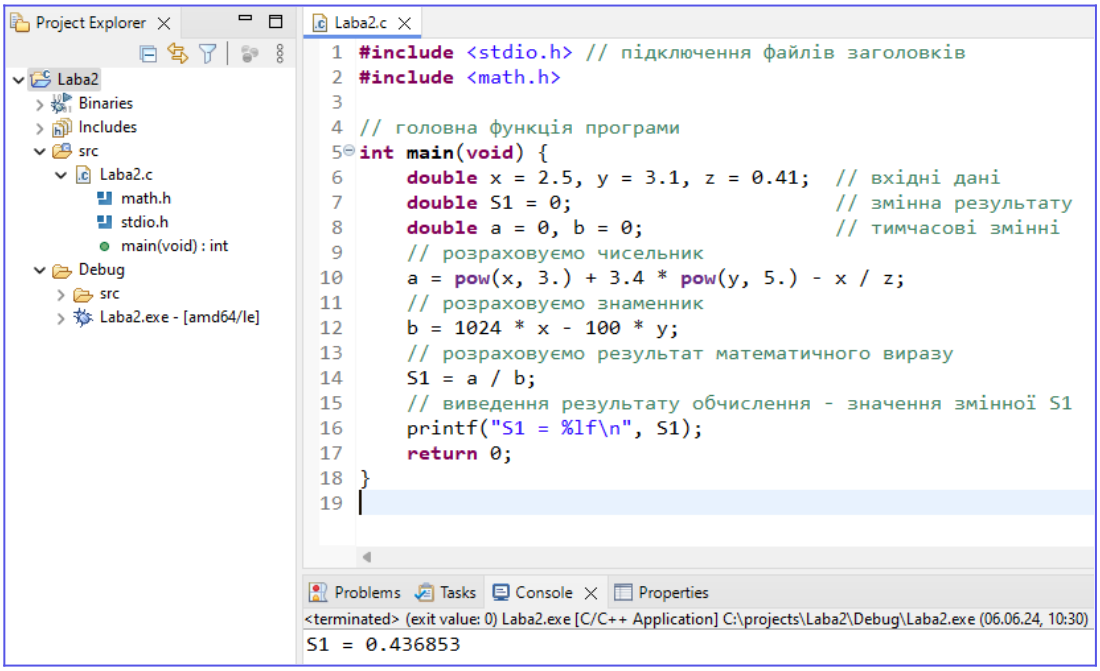


Рис. 2.2. Знімок з екрану коду програми розрахунку (2.1) з виведеним результатом обчислення

Примітка 2.2: код програми виконаний у середовищі Eclipse IDE.

Завдання

Використовуючи отримані знання, написати програму обчислення математичного виразу мовою С. Математичний вираз обрати з таблиці 1.1 першої роботи, згідно з раніше отриманим варіантом. При реалізації програми використовувати схему алгоритму, розроблену раніше. Якщо схема алгоритму раніше містила помилки або була недосконалою, то навести більш актуальну схему алгоритму, за якою побудований код програми.

Для змінних *x*, *y*, *z* з математичних виразів із таблиці 1.1 взяти такі значення:

```
double x = 2.5;
double y = 3.1;
double z = 0.41;
```

У звіт з виконаної роботи включити код програми з коментарями, опис функцій стандартної бібліотеки мови С (якщо такі є в коді для обчислення математичного виразу) та результат роботи програми. Правила оформлення звіту подані у додатку А.

При захисті роботи володіти знаннями для відповіді на питання викладача за матеріалами лекцій та теоретично-практичної частини роботи.

Робота №3.
Програмування алгоритмів розгалуження мовою С

Мета роботи: закріпити знання та навички з розробки простих консольних програм мовою програмування С та використання оператора *if*. Закріпити навички створення власних функцій.

Теоретично-практична частина

Розглянемо побудову алгоритму з розгалуженням для обчислення математичного виразу:

$$S2=\frac{x^3+3{,}4\,y^5-x/z}{1024\,x-100\,y} \qquad (3.1)$$

Алгоритми з розгалуженням містять різні умови. В залежності від вирішення цих умов, виконання алгоритму може йти різними шляхами (різними гілками).

Наприклад, якщо маємо розрахунок математичного виразу, який містить знаменник, то потрібно передбачити обчислення цього виразу тільки у разі, коли знаменник не буде дорівнювати нулю. В протилежному випадку проведення обчислення є неможливим і потрібно вивести певне повідомлення для користувача та завершити виконання програми.

Іншим типовим прикладом є використання певних математичних функцій, які можуть приймати значення тільки з передвизначеного діапазону. Так, наприклад, функція обчислення кореня квадратного не може розраховувати корінь квадратний від від'ємного значення числа (https://www.google.com/search?q=graph+y+%3D+sqrt(x)). Іншим прикладом є функція обчислення натурального логарифму, яка проводить обчислення тільки у випадку значень, які є більшими за нуль (https://www.google.com/search?q=graph+y+%3D+ln(x)).

Також в алгоритмах можуть зустрічатися різні умови, в залежності від вирішення яких, алгоритм повинен виконувати різні етапи процесів обробки, візуалізації або аналізу даних, виконання або невиконання певних дій.

Схема алгоритму для обчислення (3.1) представлена на рис. 3.1. Як бачимо в (3.1) присутні два знаменники. Для отримання значення другого знаменника (*1024x – 100y*), потрібно провести окремий математичний розрахунок. У подібному випадку краще відразу розрахувати такий знаменник та використовувати результат його обчислення як в умові при перевірці на нуль, так і для подальшого розрахунку (3.1).

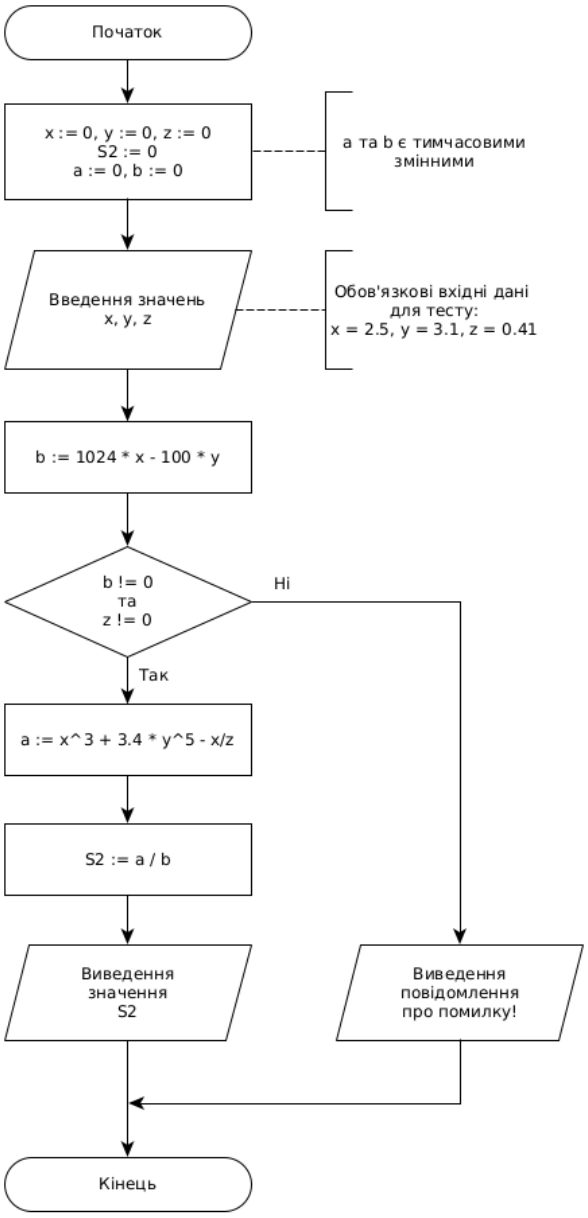


Рис. 3.1. Схема алгоритму з розгалуженням для обчислення математичного виразу (3.1)

Розрахунок виразу (3.1) можливий тільки коли значення обох знаменників (результат обчислення: *1024x – 100y* та значення змінної *z*) завжди будуть не дорівнювати нулю (рис. 3.1). В схемі на рис. 3.1 використані символи “!=” із мови програмування С – операція “не дорівнює”.

Зверніть увагу, що сформована умова є складною. Тобто складається з двох частин, які об’єднуються логічною зв’язкою “та”. Умова буде цілком вірною, коли перша частина умови та друга її частина будуть вірними (значення *b* не дорівнює нулю та значення *z* не дорівнює нулю).

Оскільки значення вхідних змінних *x*, *y*, *z* можуть бути задані користувачем будь-якими, то щоб організувати введення значень з консолі та передачу їх у відповідні змінні, можна використати функцію стандартної бібліотеки мови С, яка має ім'я `scanf()` (або іншу – `scanf_s()`), заголовок якої знаходиться у файлі *stdio.h* (додаток Б, таблиця Б.1).

Примітка 3.1: при використанні середовища розробки MS Visual Studio та стандартної бібліотеки мови С від Microsoft, для виклику класичних С99-функцій, таких, як `scanf()`, потрібно використовувати перед початком коду макрос `_CRT_SECURE_NO_WARNINGS`.

Примітка 3.2: станом на 2025 рік, функція `scanf_s()` не входить до складу стандартної бібліотеки мови С у складі компіляторів GNU GCC та Clang (для GNU/Linux-сумісних або Unix-подібних ОС). Її реалізації існують, наприклад, в бібліотеці мови С від компанії Microsoft, а також від проєкту MinGW-w64 (для ОС MS Windows).

Таким чином, згідно схеми алгоритму, маємо наступний фрагмент коду мовою С:

```
#define _CRT_SECURE_NO_WARNINGS
. . .
double x = 0, y = 0, z = 0;   // змінні для збереження вхідних даних
double S2 = 0;                // змінна для збереження результату
double a = 0, b = 0;          // тимчасові змінні

// організовуємо запит для користувача
// на введення значень x, y, z
printf("Input x, y, z values: ");
fflush(stdout);
// вводимо значення вхідних даних
scanf("%lf%lf%lf", &x, &y, &z);
// або використати scanf_s() для певних редакцій бібліотек (див. примітки)

// розраховуємо другий знаменник
b = 1024 * x - 100 * y;

// чи значення b не дорівнює нулю та
// значення z не дорівнює нулю?
if ((b!=0)&&(z!=0)) { // якщо "Так"
      // розраховуємо чисельник з врахуванням значення першого знаменника
      a = pow(x, 3.) + 3.4 * pow(y, 5.) - x / z;
      // розраховуємо результат математичного виразу
      S2 = a / b;
      // виведення результату обчислення – значення змінної S2
      printf("S2 = %lf\n", S2);
}
// інакше, якщо "Ні" - виводимо повідомлення про помилку
else printf("ERROR of calculation!\n");
. . .
```

Формуємо окремий файл коду мовою С (наприклад, *Laba3_1.c*) з врахуванням підключення потрібних файлів заголовків за допомогою директиви препроцесора *include*, та включаємо цей код в реалізацію головної функції програми (головної точки входу в програму) – в функцію `main()`.

Завершений код програми розрахунку (3.1) представлений на рис. 3.2.

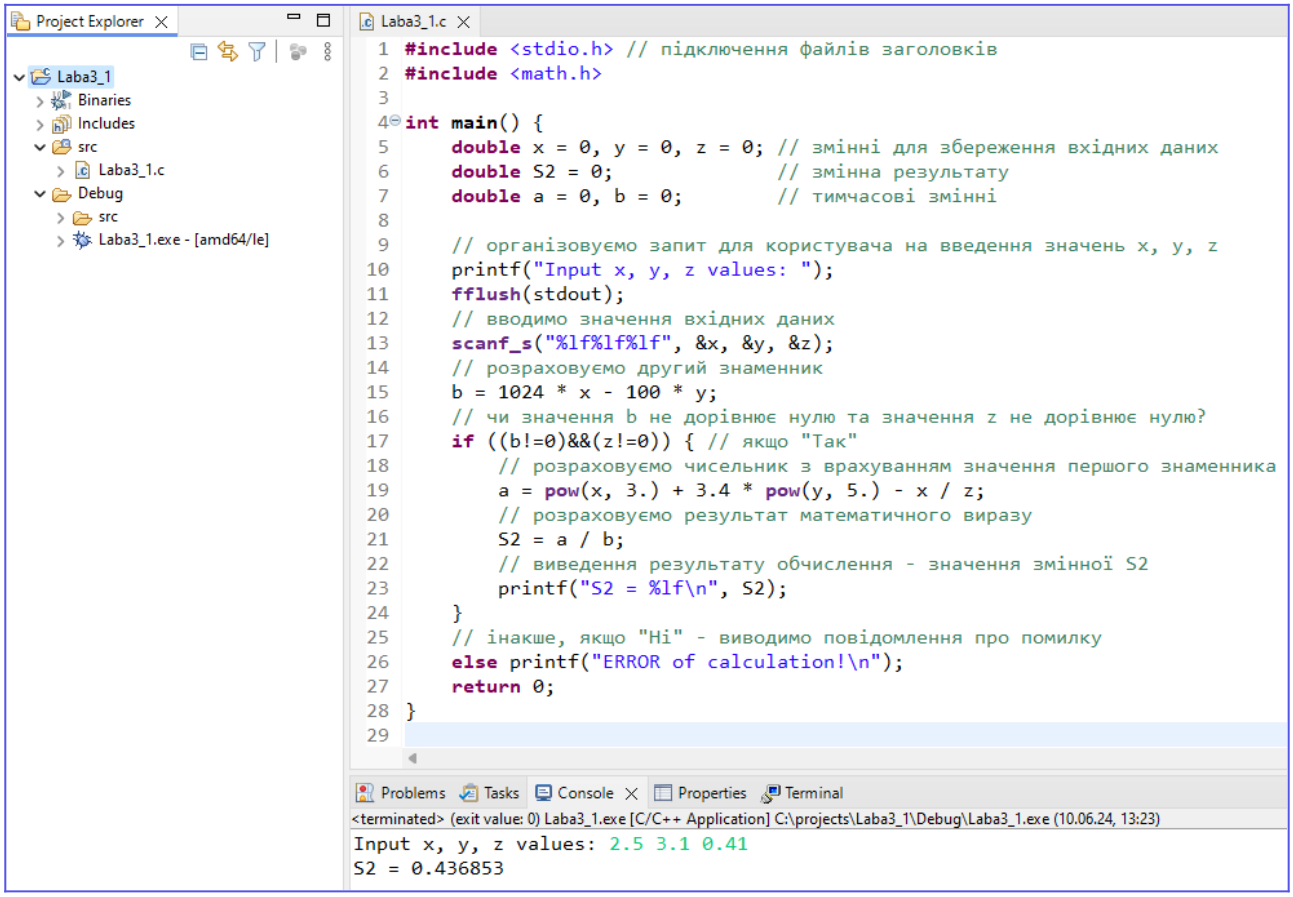


Рис. 3.2. Знімок з екрану коду програми розрахунку (3.1) з виведеним результатом обчислення у разі введення тестових значень

Примітка 3.3: код програми виконаний у середовищі Eclipse IDE під платформу MS Windows з використанням компілятору та бібліотеки проєкту MinGW-w64 та є сумісним для використання у середовищі MS Visual Studio. У разі створення коду не в ОС MS Windows, виклик функції `scanf_s()` можна замінити на `scanf()`.

Перед введенням значень змінних *x*, *y*, *z* для користувача треба сформувати запит, в якому вказати, що конкретно йому треба зробити й для чого. Це повідомлення можна вивести за допомогою функції `printf()`.

При використанні зв'язки функцій *printf-scanf* (або їх різновидів) може виникнути ситуація, яка пов'язана з їх некоректною роботою в певних консолях. Не завжди, але ця некоректна робота може проявлятися в тому, що рядок

повідомлення з функції `printf()` не виводиться на консоль, а відразу буде очікуватися введення даних для `scanf()/scanf_s()`. Це можна спостерігати, наприклад, при роботі з консоллю в Eclipse IDE для MS Windows. Тому радять для вірного відпрацювання зв'язки цих функцій, перед викликом функції `scanf()/scanf_s()`, викликати функцію стандартної бібліотеки мови С, яка очищує буфер потоку виведення – функцію `fflush()` (додаток Б, таблиця Б.1) – рядок 11 (рис. 3.2).

Досить часто виконання задачі розбивають на підзадачі – виносять окремі частини програмного коду у власні функції. Також у функції виносять частини коду, який може повторюватися декілька разів в різних місцях за побудованим алгоритмом.

Функція – це сукупність оголошень та операторів, призначена для виконання деякої окремої задачі.

Оголошення функції специфікує ім'я, формальні параметри та завжди закінчується символом ';'. Допускається відсутність імен формальних параметрів в оголошенні функції.

Визначення функції в програмі (її реалізація) має бути тільки одне. Воно специфікує ім'я, формальні параметри та тіло функції.

Якщо функція не повертає в точку виклику значень, то тип повернення функції вказується ключовим словом *void*.

Для повернення значення певного типу з функції використовують оператор *return*.

Досить часто виникає ситуація, коли з функції потрібно повернути декілька значень. Для цього можна використовувати механізм покажчиків (вказівників) на потрібний тип даних, які вказуються в параметрах функції.

Наприклад, нехай потрібно розробити функцію, яка буде обслуговувати введення вхідних даних та повертати значення типу *bool*: `true` – якщо введення даних відбулося або повертати `false` у разі невдачі. Приклад реалізації такої функції представлений на рис. 3.3.

В цьому прикладі використаний макрос `bool`, який обраний для сумісності з мовою C++. Насправді до редакції мови C23, починаючи з C99, для роботи з булівським типом даних використовувався тип `_Bool`. Для сумісності з синтаксисом C++ було введено окремий файл *stdbool.h*, в якому вже визначені макроси `bool`, `true` та `false`. Починаючи з редакції C23, в мову введений тип даних `bool` та константи `true` та `false`. Це дозволяє вже не використовувати додатково файл *stdbool.h*.

Слід зауважити, що стандарт C23 станом на початок 2025 року підтримується ще не всіма компіляторами. Тому можна користуватися прийнятим у C99 варіантом (рис. 3.3).

В рядку 5 (рис. 3.3) подано *оголошення* функції, на на рядках 18 – 24 дано *визначення* функції (її реалізація).

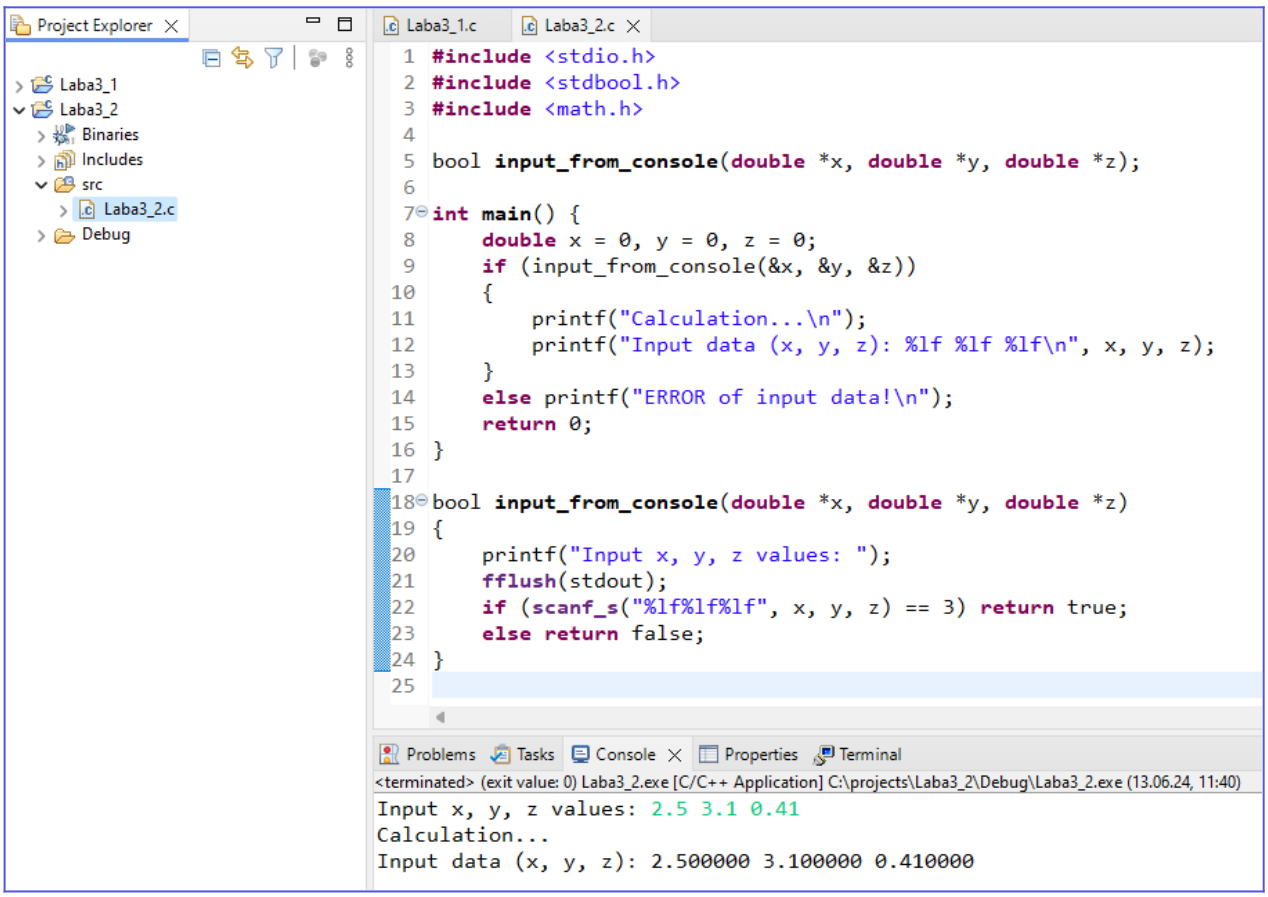


Рис. 3.3. Приклад реалізації функції введення даних з консолі
та її виклику з основної функції програми

З прикладу на рис. 3.3 бачимо, що якщо функція `scanf_s()` повертає значення 3 (рядок 22), то це означає, що були введені коректно значення для трьох змінних, адреси яких передані в функцію `scanf_s()`. Виклик функції відбувається в рядку 9 головної функції програми. Саме головна функція `main()` вже вирішує, як далі будуть розгортатися події: чи проводити далі розрахунки, чи завершити роботу програми у разі невірного введення вхідних даних.

Таким чином, розрахунок виразу (3.1) може бути розділений на декілька подібних функцій: в першій функції буде розрахований чисельник, друга функція буде розраховувати знаменник, а третя функція буде розраховувати фінальний вираз. При побудові першої та третьої функцій потрібно також враховувати умови для вірного обчислення (тобто в даному випадку робити перевірку на нуль). Друга функція буде повертати просто результат розрахунку знаменника.

Зверніть увагу, що при побудові функцій, які в середині будуть використовувати умову(и), потрібно повертати результат загальної роботи

функції, наприклад, через оператор *return*, а результат обчислень повертати через покажчик(и), як це було продемонстровано вище на прикладі (рис. 3.3). В прикладі результати розміщувалися за адресами, які були передані аргументами в функцію при її виклику в рядку 9. Подібним способом можна організувати повернення результату обчислення через параметр-покажчик.

Завдання

1. Використовуючи знання, отримані на лекціях та з теоретично-практичної частини роботи, написати мовою програмування С програму обчислення математичного виразу з таблиці 1.2 першої роботи згідно раніше отриманого варіанту. При реалізації програми використовувати схему алгоритму, що розроблений раніше. Якщо схема алгоритму раніше містила помилки або була недосконалою, то навести більш актуальну схему алгоритму, за якою буде побудований код програми.

2. Написати другу версію програми з пункту 1 цього завдання, використовуючи механізм функцій – гілки алгоритму для обчислення різних частин математичного виразу повинні бути оформлені у вигляді функцій.

Якщо функція повинна містити перевірку(и) перед обчисленням, то така функція в якості результату повертає значення типу *bool* (обчислення виконано (*true*), чи не виконано (*false*)), а сам результат обчислення повертається через параметр-покажчик.

У разі коли функція проводить просте обчислення за алгоритмом лінійної структури, то вона відразу повертає результат обчислення.

3. Запропонувати різні варіанти тестових вхідних даних, які покажуть вірне виконання функцій другої програми. До таких тестових даних обов’язково включити дані з роботи №2.

У звіт до роботи включити коди розроблених програм з коментарями та результати їх роботи. Правила оформлення звіту подані в додатку А.

При захисті роботи володіти знаннями для відповіді на питання викладача за матеріалами лекцій та теоретично-практичної частини роботи.

Робота №4.
Програмування циклічних алгоритмів мовою С

Мета роботи: закріпити знання з розробки простих консольних програм мовою програмування С та використання оператора циклу *for*.

Теоретично-практична частина

Розглянемо побудову циклічного алгоритму для розрахунку математичного виразу:

$$S3=\sum_{i=1}^{11}\sin a_i^2 \qquad (4.1)$$

В циклічному алгоритмі певні кроки повторюються вказану кількість разів. Якщо цикл є нескінченним, то все одно повинен бути крок, на якому за певних умов його робота буде завершена.

В математичному виразі (4.1) для розрахунку суми, можна використовувати циклічний алгоритм – на кожному кроці виконується розрахунок функції *sin* та результат її обчислення додається до попереднього результату розрахунку. Всі ці дії повторюються послідовно за вказаною кількістю кроків (у випадку виразу (4.1) – 11 разів). В якості вхідних даних алгоритм передбачає певну множину значень a_i загальною кількістю 11.

Приклад схеми алгоритму розрахунку (4.1) представлений на рис. 4.1. Слід зазначити, що подібний алгоритм розрахунку можна будувати, виходячи з різних міркувань та умов програми.

Наприклад, якщо треба зберігати вхідну множину даних a_i, для її подальшого використання, то потрібно передбачити в програмі відповідну структуру даних. В мові С це одновимірний масив значень певного типу.

Оскільки ***індексація елементів масивів в мові С починається з нуля***, то це знайшло відображення на подальших схемах алгоритмів, представлених на рис. 4.1 та 4.2. Загальна кількість вхідних даних при цьому все також дорівнює 11-ти. В схемах алгоритмів ця загальна кількість за виразом (4.1) контролюється.

Якщо в програмі не вимагається окреме збереження значень a_i, то такі значення можуть бути введені користувачем та негайно використані для обчислення виразу. Така реалізація відображена на рис. 4.2.

Зверніть увагу, що самі цикли можуть бути описані в схемах та побудовані в програмах теж по різному. Тобто можна використовувати конструкції для побудови циклів з передумовою (рис. 4.2, а), з післяумовою (рис. 4.2, б), а також будувати ітераційні цикли з лічильником (рис. 4.2, в).

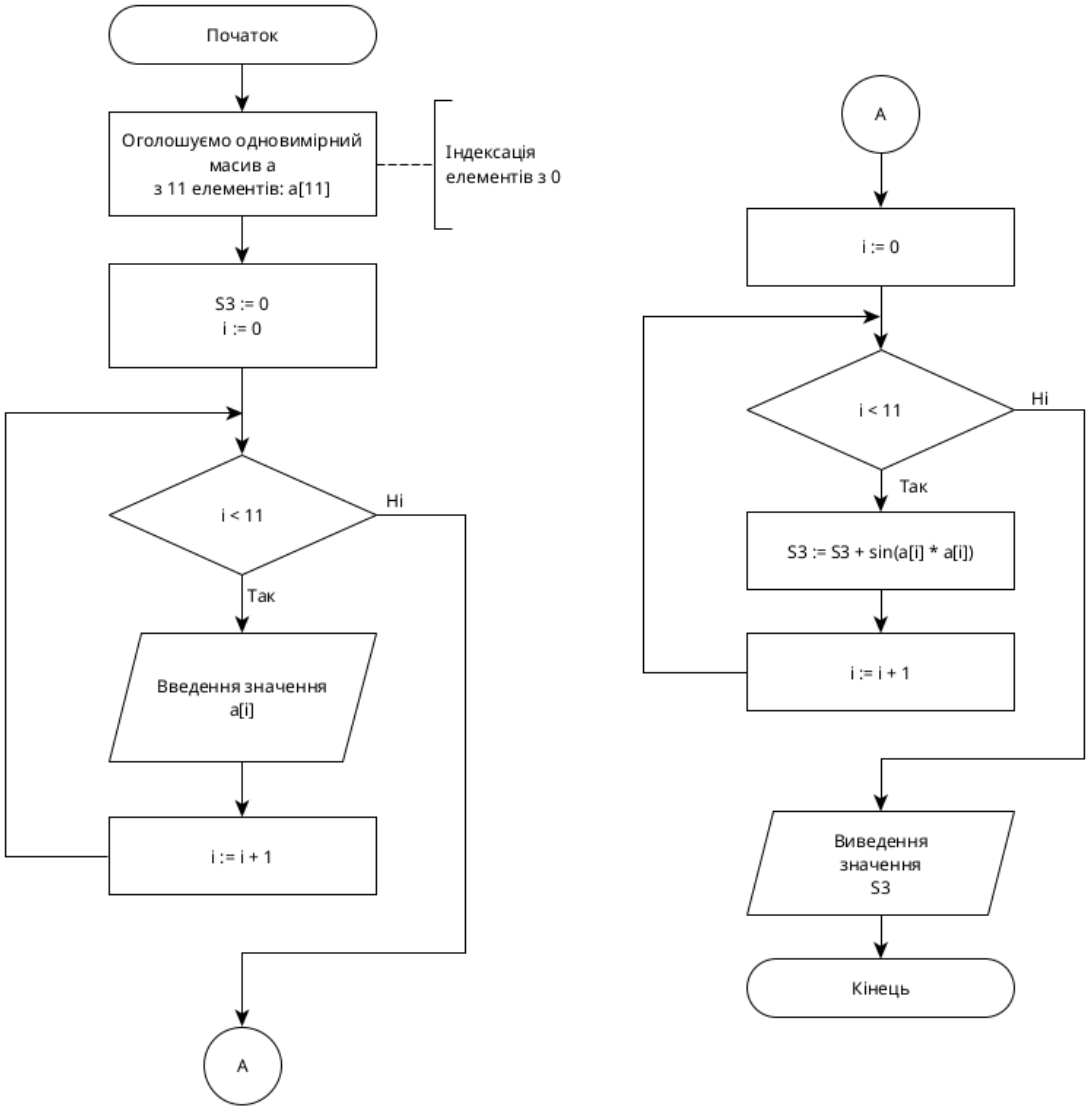


Рис. 4.1. Схема алгоритму обчислення математичного виразу (4.1) на основі циклів з передумовою та із збереженням вхідних даних в окремому одновимірному масиві

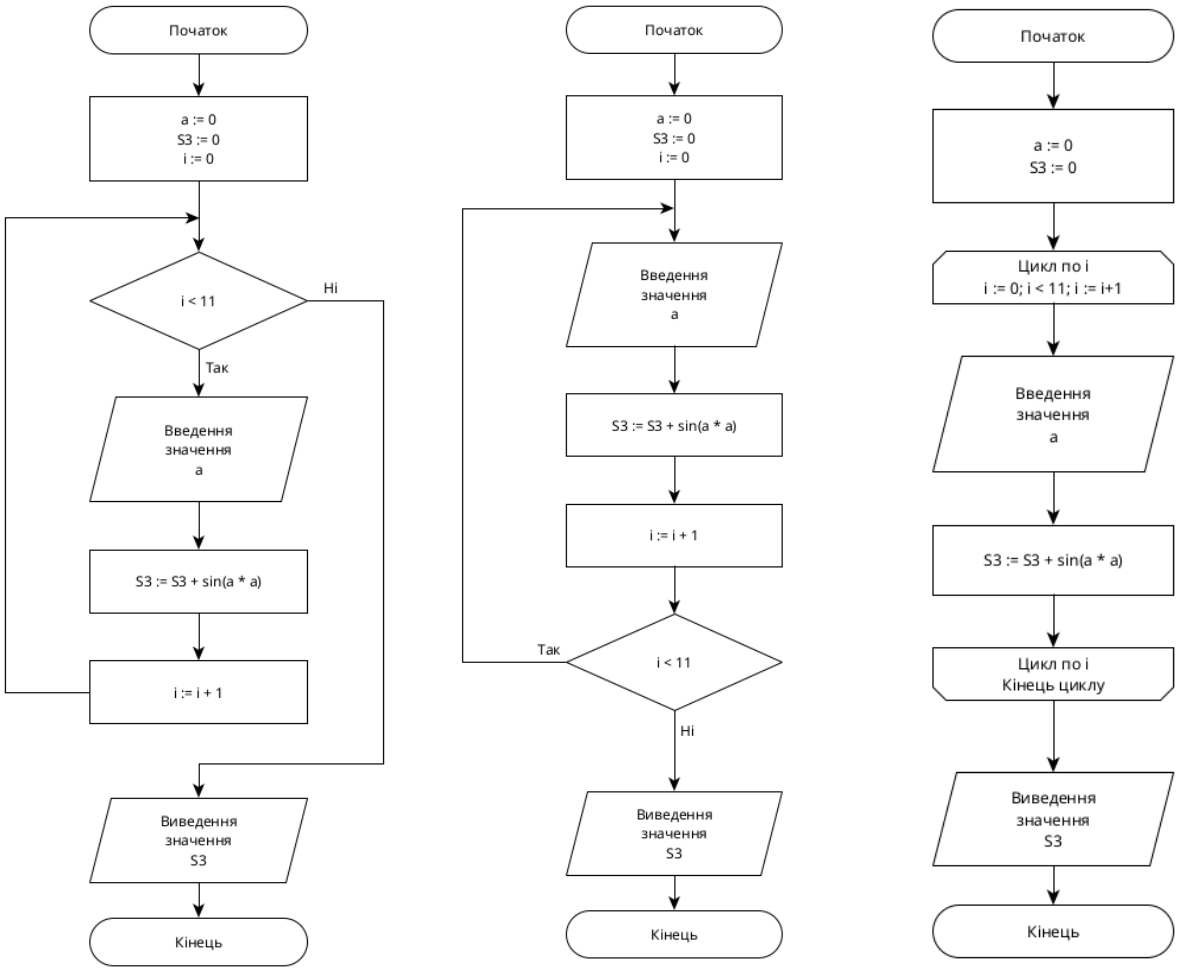
При будуванні будь-яких циклів потрібно визначити початкові умови, за якими розпочнеться цикл, а також умови за якими він завершиться. Так при розрахунку (4.1) повинна бути змінна, яка контролює доступ до певного елементу вхідних даних, а також рахує крок виконання ітерації циклу. Такі змінні називають лічильниками. Типовим іменем для подібної змінної є ім’я: i. Часто змінна-лічильник може виступати також індексом елемента масиву (рис. 4.1).

Також в даному випадку потрібно визначити початкове значення суми. Якщо на кожній ітерації циклу буде виконано операцію додавання результату обчислення функції *sin* до попереднього результату додавання, то виникає питання: до якого результату потрібно буде додати перший результат

розрахунку функції *sin*? В такій ситуації початковим значенням суми буде значення нуль (*S3* привласнюємо 0).

Умовою, за якою цикл буде завершений, є досягнення значення лічильника 10-ти (при умові, що початкове значення дорівнювало нулю – рис. 4.1 та 4.2).

Зверніть увагу, що у схемі алгоритму з рис. 4.2, в), використані символи циклу, що є рекомендованими в Національному стандарті України ДСТУ ISO 5807:2016 – спеціальні графічні символи початку та завершення циклу.



а) б) в)

Рис. 4.2. Приклади схем алгоритмів обчислення математичного виразу (4.1) без збереження вхідних даних в окремому одновимірному масиві:
а) – на основі циклу з передумовою; б) – на основі циклу з післяумовою;
в) – на основі циклу з лічильником

Таким чином, основну частину алгоритму з рис. 4.1 можна представити наступним фрагментом коду мовою С:

```
. . .
double a[11] = {0};
double S3 = 0;
for (int i = 0; i<11; i++) {
      printf("Enter a[%d]: ", i+1);
      fflush(stdout);
      scanf_s("%lf", &a[i]);
}

for (int i = 0; i<11; i++) S3 += sin(a[i]*a[i]);
printf("S3 = %lf\n", S3);
. . .
```

Далі формуємо окремий файл коду мовою С (наприклад, *Laba4_1.c*) з врахуванням підключення потрібних файлів заголовків за допомогою директиви препроцесора *include*, та включаємо цей код в реалізацію головної функції програми (в головну точку входу в програму) – в функцію `main()`.

Завершений код програми розрахунку (4.1) представлений на рис. 4.3.

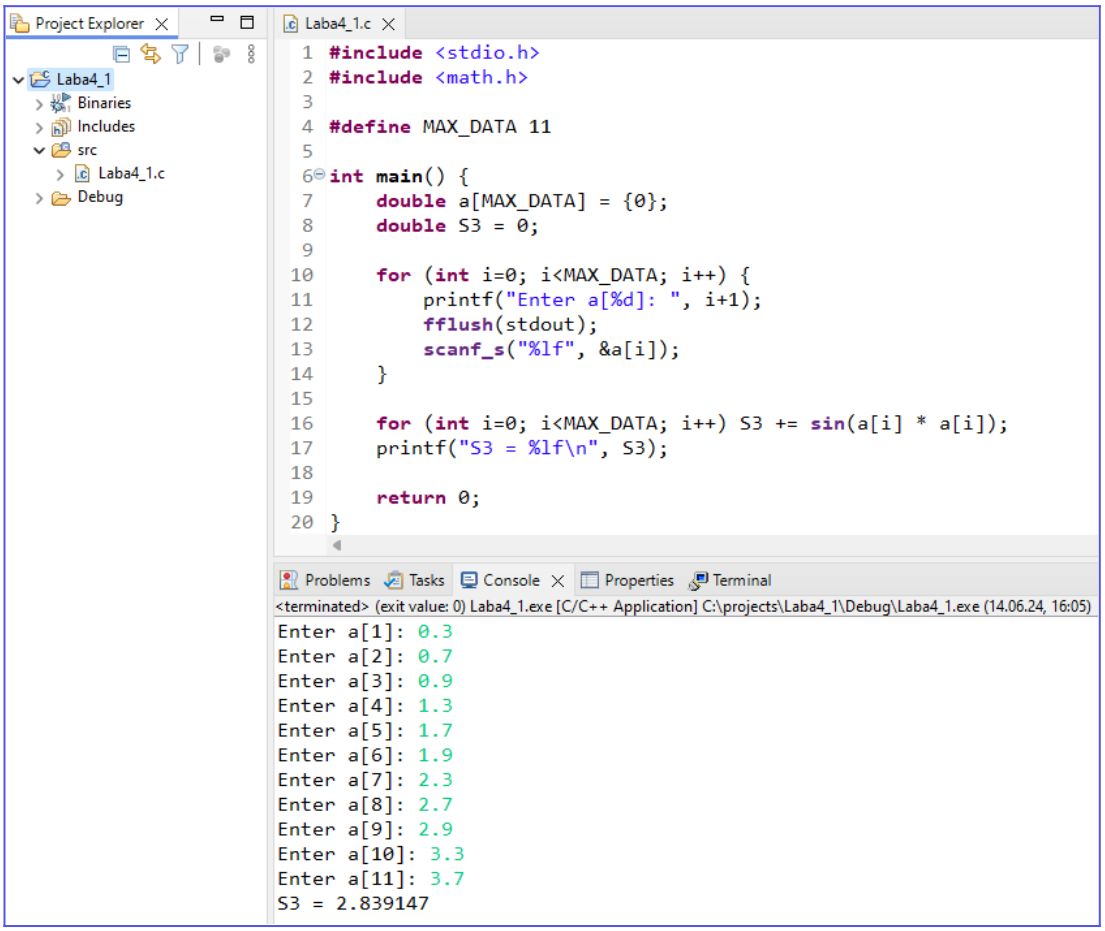
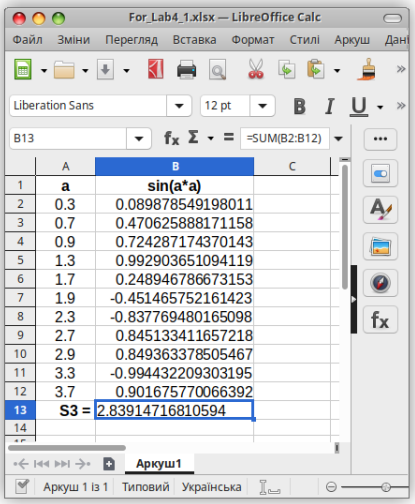


Рис. 4.3. Знімок з екрану коду програми розрахунку (4.1) з виведеним результатом обчислення у разі введення певних тестових значень

Модифікуємо програмний код з врахуванням вже передвизначеної тестової послідовності значень a_i. Нехай вже відомий результат обчислення за передвизначеною послідовністю з використанням іншої програми розрахунку. Наприклад, MS Office Excel або LibreOffice Calc (рис. 4.4).



	A	B	C
1	a	sin(a*a)	
2	0.3	0.089878549198011	
3	0.7	0.470625888171158	
4	0.9	0.724287174370143	
5	1.3	0.992903651094119	
6	1.7	0.248946786673153	
7	1.9	-0.451465752161423	
8	2.3	-0.837769480165098	
9	2.7	0.845133411657218	
10	2.9	0.849363378505467	
11	3.3	-0.994432209303195	
12	3.7	0.901675770066392	
13	S3 =	2.83914716810594	
14			

Рис. 4.4. Результат розрахунку (4.1) на тестовому наборі даних в програмі LibreOffice Calc

Нехай маємо спеціальний макрос `USE_TEST`, який буде керувати збіркою коду: коли він доступний для препроцесора мови С, то введення з консолі в програмі не буде відбуватися та вектор з одинадцяти елементів буде мати певні тестові значення. Така модифікація коду має вигляд:

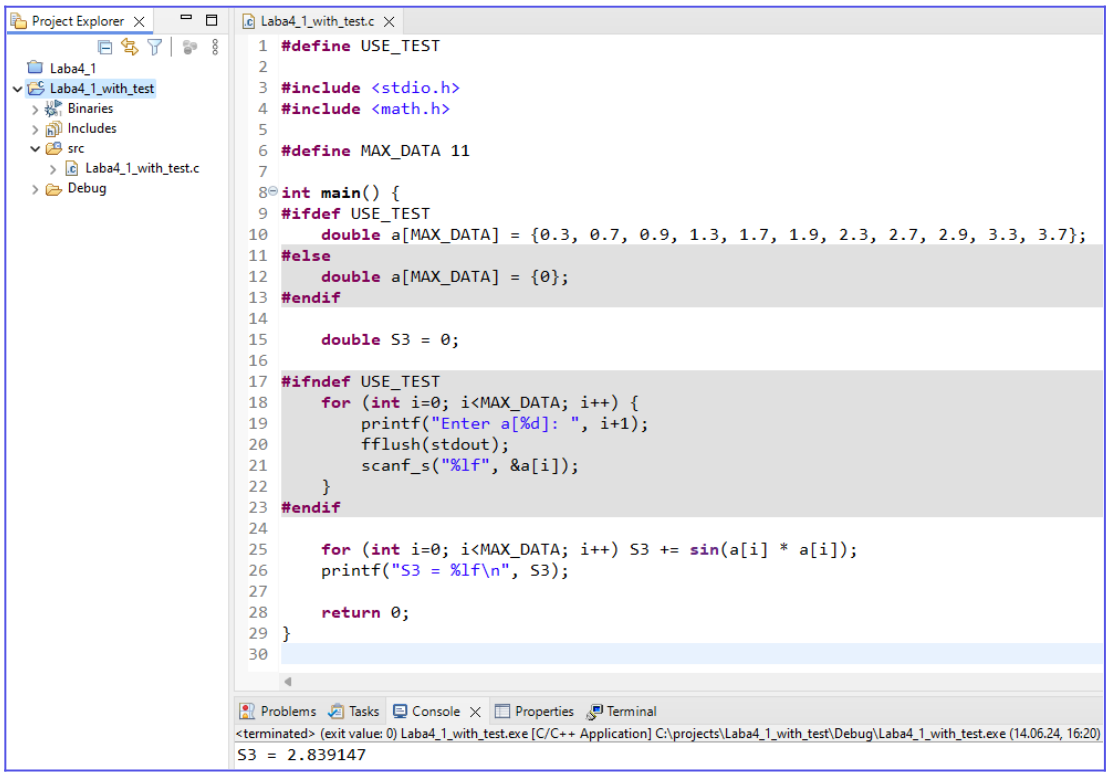


Рис. 4.5. Перше вдосконалення програми розрахунку (4.1)

Вірною технікою при виконанні тестувань є залучення певних стандартних бібліотек для спрощення процедури автоматизованого модульного тестування. В цьому простому випадку можна додати свою власну функцію, яка буде відпрацьовувати процедуру перевірки значення *S3* після розрахунку на співпадіння з еталонним значенням, яке заздалегідь відомо для тестового набору даних і знайдено при обчисленні в іншій програмі або розраховано власноруч. Нехай для порівняння створимо функцію з наступним кодом:

```
31 void assert_equal_double(double expected, double actual, int precision)
32 {
33 #ifdef USE_TEST
34     double v = trunc(actual * pow(10, precision))/pow(10, precision);
35     if (v == expected) {
36         fprintf(stderr, "TEST OK! Expected == Actual == %.15g\n", expected);
37         fflush(stderr);
38     }
39     else {
40         fprintf(stderr, "TEST Fail! Expected %.15g != Actual %.15g\n", expected, v);
41         exit(1);
42     }
43 #endif
44 }
```

Рис. 4.6. Тестова функція порівняння двох значень
з плаваючою комою певної точності

Функція `assert_equal_double()` приймає три аргументи: *expected* – це значення, яке очікується (тобто значення, яке заздалегідь вже відомо), *actual* – це значення, яке отримане в результаті розрахунку та *precision* – це точність, з якою потрібно провести порівняння. Точність – це кількість розрядів після коми, які будуть брати участь у порівнянні.

В даному випадку в якості значення *expected* буде виступати значення: *2,839147*. Тобто значення з точністю до шостого знаку.

Для округлення актуального значення обираємо техніку множення на 10 в певному ступені, далі робимо відкидання дробової частини та знову повернення у значення з плаваючою комою діленням на 10 у визначеному ступені. Функція `trunc()` оголошена в *math.h* (додаток Б, таблиця Б.5). Після цього у рядку 35 робиться порівняння. Якщо результат порівняння хибний, то програма буде завершена через виклик функції `exit()` із сповіщенням, що тест провалений. Функція `exit()` оголошена в *stdlib.h* (додаток Б, таблиця Б.2).

Сповіщення виводяться в стандартний потік виведення для помилок *stderr* за допомогою функції форматованого виведення `fprintf()`, що оголошена в *stdio.h* (додаток Б, таблиця Б.1).

Таким чином, завершена програма розрахунку (4.1) з тестувальною частиною представлена на рис. 4.7.

Актуальне значення визначене в програмі як константа з плаваючою комою в рядку 14, а в рядку 31 йде виклик тестової функції (рис. 4.7) з вказівкою точності порівняння до шостого знаку після коми.

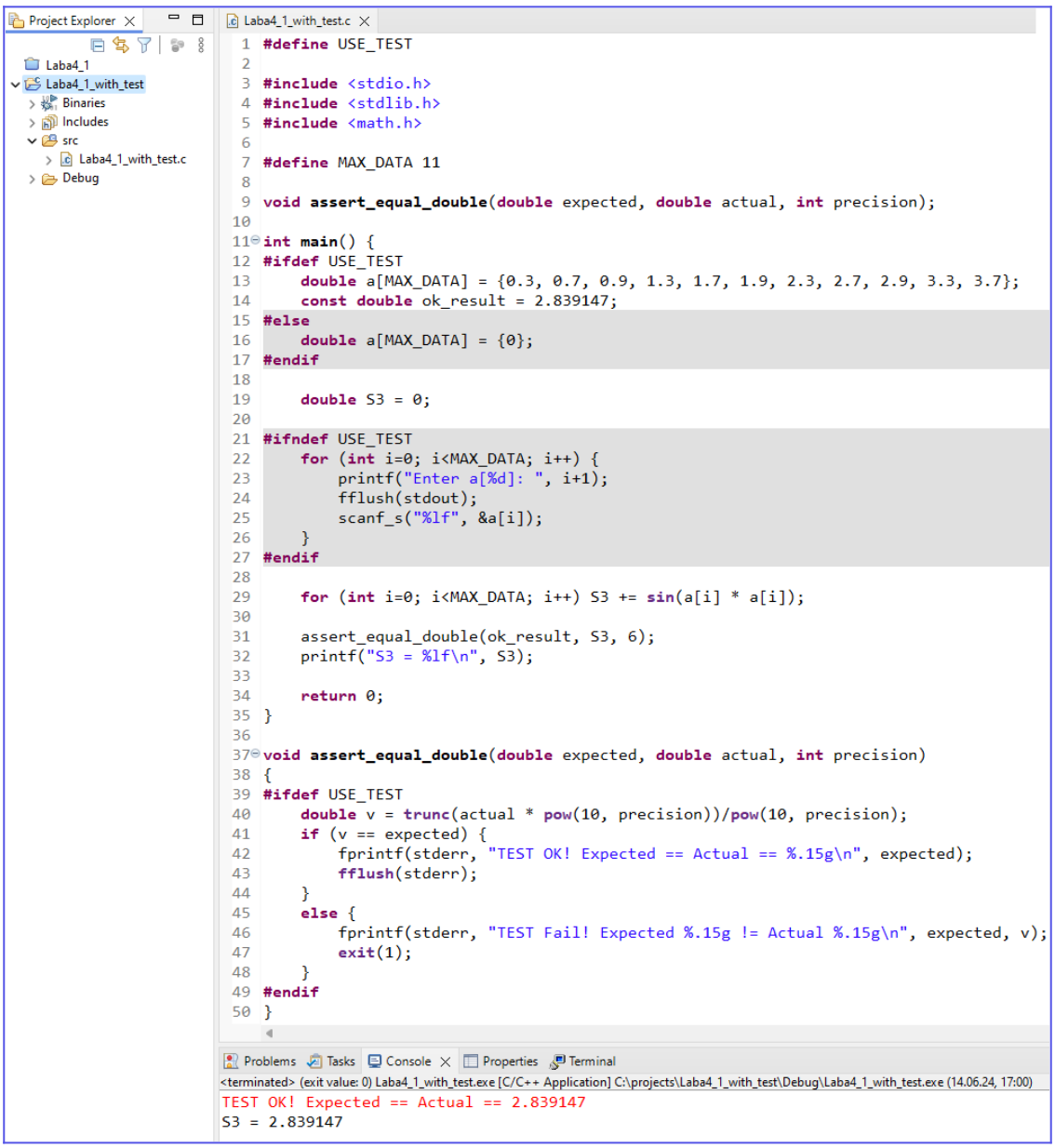


Рис. 4.7. Друге вдосконалення програми розрахунку (4.1)
з вдалим результатом тестування

Якщо в 31 рядку змінити, наприклад, точність, то тест із передвизначеним результатом буде провалено:

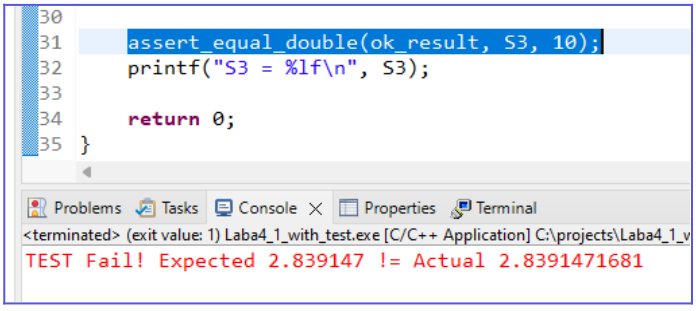


Рис. 4.8. Результат проваленого тесту

Таким чином, якщо був отриманий завчасно результат розрахунку математичного виразу, створений алгоритм розрахунку та програма мовою С, але тест з вказаною точністю не пройшов, то це свідчить про помилки при описі розрахунку математичного виразу мовою С або про помилки в самому алгоритмі розрахунку, або про помилки в реалізації тестової функції чи невірно вказаної точності результату для порівняння.

Вхідні дані для консольних програм розрахунків можна вводити з консолі, можна зчитувати з файлів або передавати в якості аргументів програми при її запуску з командного рядку. Розглянемо більш детальніше останній варіант.

Основною точкою входу в програму є функція `main()`. Її заголовок може бути різним, в залежності від задачі, що вирішується. Приклади варіантів заголовків функції `main()`:

```
int main()

void main()

// функція main() з двома параметрами
int main(int argc, char *argv[])

// функція main() з трьома параметрами
int main(int argc, char *argv[], char *envp[])
```

Як правило параметри *argc* та *argv* призначені для обробки даних, які надходять до програми у вигляді аргументів, що передані програмі через командний рядок.

Параметр *envp* використовується тільки у випадках, коли програмі необхідно обробити дані, що зберігаються в так званих змінних середовища (environment variables) ОС. В різних ОС значення в *envp* можуть сильно відрізнятися.

Таким чином, для передачі значень для розрахунку можна використати варіант функції `main()` з двома параметрами.

Параметр *argc* містить кількість аргументів, включаючи назву програми, що викликана в командному рядку (елемент *argv[0]*). Через параметр *argv[i]* можна звернутися до *i*-го аргументу програми. За правилами індексація в С-масивах починається з нуля. У останнього елемента масиву *argv* буде індекс *argc-1*. *argv[1]* буде містити перший аргумент програми, який переданий в програму через командний рядок.

Зверніть увагу, що аргументи програми, які передані програмі через командний рядок, передаються в функцію `main()` як рядки. Тобто, якщо ці аргументи повинні інтерпретуватися як значення цілого типу або як значення з

плаваючою комою, то потрібно додатково викликати функції-конвертори у відповідний тип даних. Наприклад, можна скористатися функціями `atoi()`, `atof()` або подібними (додаток Б, таблиця Б.2).

Приклад фрагменту коду з обробкою вхідних даних для розрахунку математичного виразу (4.1) із значеннями, які передані як аргументи програми через командний рядок та потрапляють у функцію `main()` через параметр *argv*:

```
. . .
if (argc == (MAX_DATA+1)) {
      for (int i=0; i<MAX_DATA; i++) {
            a[i] = atof(argv[i+1]);
            . . .
      }
      . . .
}
else printf("ERROR in program input parameters!\n");
. . .
```

Задати аргументи програми можна або в середовищі розробки (якщо таке передбачено інтерфейсом IDE), або напряму у вікні терміналу консолі. Наприклад, в Eclipse IDE для цього можна обрати пункт меню *Run \ Profile Configurations* та у діалоговому віконці перейти до вкладки *Arguments* (рис. 4.9). Аргументи програми вводять через пробіл.

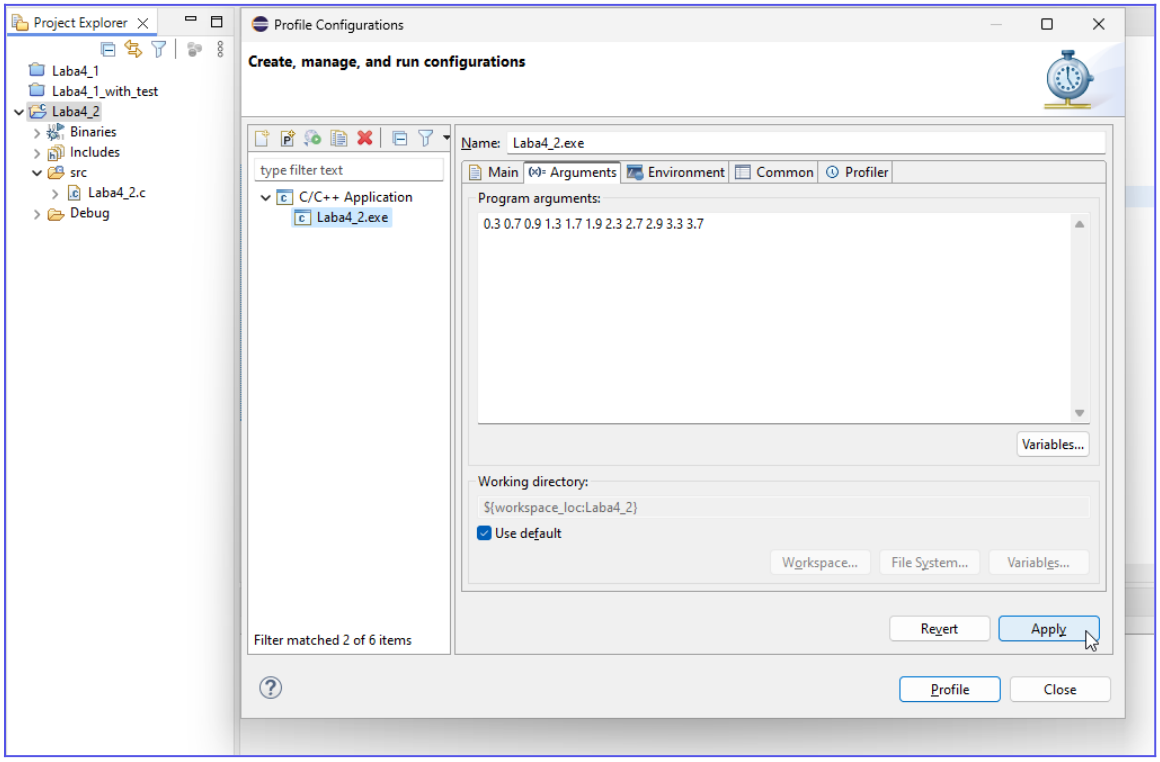


Рис. 4.9. Вікно встановлення аргументів програми в середовищі Eclipse IDE

На рис. 4.10 показаний приклад виклику програми з командного рядку з передачею їй аргументів – тестових значень для розрахунку математичного виразу (4.1) – по факту аргументів функції `main()`.

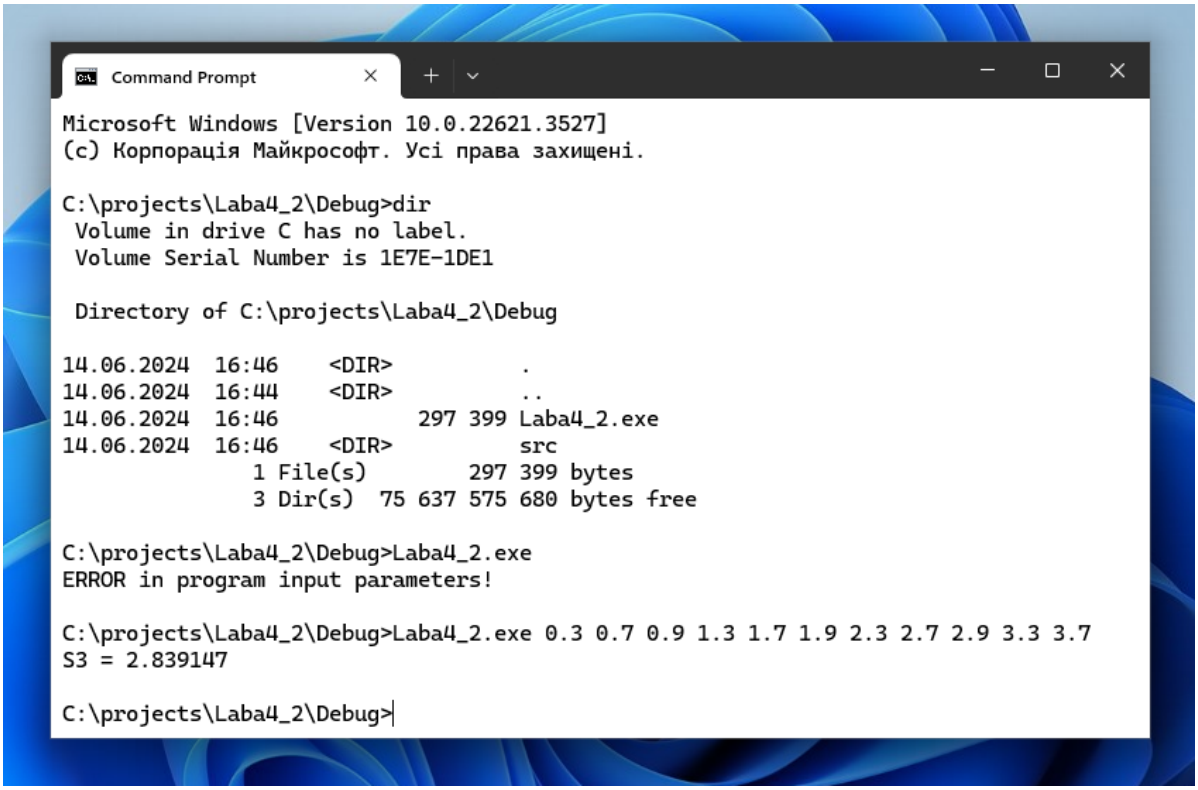


Рис. 4.10. Приклад виклику програми з аргументами в командному рядку

Слід пам’ятати, що якщо аргументи програми самі містять пробіли, то обов’язково таке значення повинно бути екрановано подвійними лапками. Наприклад:

```
C:\myprog\myprog.exe 1 2 3 "this is test string" 4 5 6 7
```

Завдання

1. Використовуючи знання, які отримані на лекціях та з теоретично-практичної частини роботи, написати мовою програмування С програму обчислення виразу з таблиці 1.3 першої роботи, згідно раніше отриманого варіанту. При реалізації програми використовувати схему алгоритму, що розроблений раніше. Якщо схема алгоритму раніше містила помилки або була недосконалою, то навести більш актуальну схему алгоритму, за якою

побудований код програми. ***Врахувати, що вхідні дані для обчислення зберігаються в одновимірному масиві***.

2. Розробити версію коду з можливістю тестування математичного виразу за варіантом. Тест проводити на векторі вхідних даних:

0.3, 0.7, 0.9, 1.3, 1.7, 1.9, 2.3, 2.7, 2.9, 3.3, 3.7

Для отримання еталонного результату розрахунку використовувати програми на кшталт MS Office Excel, LibreOffice Calc або подібні. Залучити функцію `assert_equal_double()`.

3. Розробити другу версію програми, яка використовує ***динамічний*** масив *a*. В програмі передбачити ***передачу вхідних даних через аргументи програми*** та їх подальшу обробку. Включити в другу версію програми можливість тестування на вірний результат.

У звіт до роботи включити коди розроблених програм з коментарями та результати їх роботи. Правила оформлення звіту подані в додатку А.

При захисті роботи володіти знаннями для відповіді на питання викладача за матеріалами лекцій та теоретично-практичної частини роботи.

Робота №5.
Робота з динамічними векторами та матрицями

Мета роботи: закріпити знання по роботі з динамічними масивами даних, основним алгоритмом обробки векторів та матриць, а також навички створення багатомодульних програм.

Теоретично-практична частина

Нехай потрібно організувати певні операції з множиною значень типу *double*. На момент створення коду програми невідомо, скільки значень буде залучено для обчислення, але точно відомо, що ці значення утворюють одновимірний масив даних. В цьому випадку пам’ять під збереження значень вхідних даних буде виділена динамічно. В мові С для цього існує певна множина функцій, які оголошені в файлі *stdlib.h* (додаток Б, таблиця Б.2) та використовуються змінні-покажчики (вказівники) на певний тип.

Приклад коду функції, яка дозволяє організувати виділення потрібного об’єму динамічної пам’яті та організувати введення даних з консолі:

```
. . .
// оголошуємо змінну v – покажчик на double
double *v = NULL;
. . .
// організовуємо виділення динамічної пам’яті
// під масив даних за потреби
// та введення значень
v = input_from_console(v, count);
. . .
// відпрацьовуємо з динамічним масивом даних
. . .
// повертаємо динамічну пам’ять системі
free(v);
. . .
double *input_from_console(double *data, int count)
{
   if (NULL == data) data = calloc(count, sizeof(double));

   // повторно перевіряємо, чи була виділена пам'ять
   if (NULL != data) {
      // організовуємо введення даних з консолі
      printf("Enter an array of %d elements separated by a space:\n", count);
      fflush(stdout);
      for (int i=0; i<count; i++) scanf_s("%lf", &data[i]);
   }
   return data;
}
```

Спочатку у функції йде перевірка, чи була використана динамічна пам’ять. Якщо *data* буде дорівнювати нульовій адресі, то за допомогою функції `calloc()` здійснюється спроба виділення динамічної пам’яті зазначеного об’єму. Після чого переходимо до організації введення даних з консолі.

Для зручності написання коду великих об’ємів, для покращення структуризації коду програм, певні функції, константи, директиви препроцесору, типи даних користувача прийнято відносити в окремі модулі програми.

Покажемо приклад створення багатомодульної програми. Нехай будемо оперувати функціями:

`int input_number_of_values()` – організовує введення з консолі значення кількості елементів масиву;

`double *input_from_console(double *data, int count)` – організовує введення з консолі значень з плаваючою комою одновимірного динамічного масиву та при необхідності виділення динамічної пам’яті;

`void print(const double *data, int count)` – виведення значень одновимірного динамічного масиву на консоль;

`double get_sum(const double *data, int count)` – повернення суми значень з одновимірного динамічного масиву;

`void assert_equal_double(double expected, double actual, int precision)` – функція тестування на співпадіння значень з плаваючою комою.

Оголошення перелічених функцій винесемо в окремий файл *mylib.h*, а в файл *mylib.c* винесемо їх реалізацію. Все це будемо робити у складі проєкту під назвою *TestMyLib*.

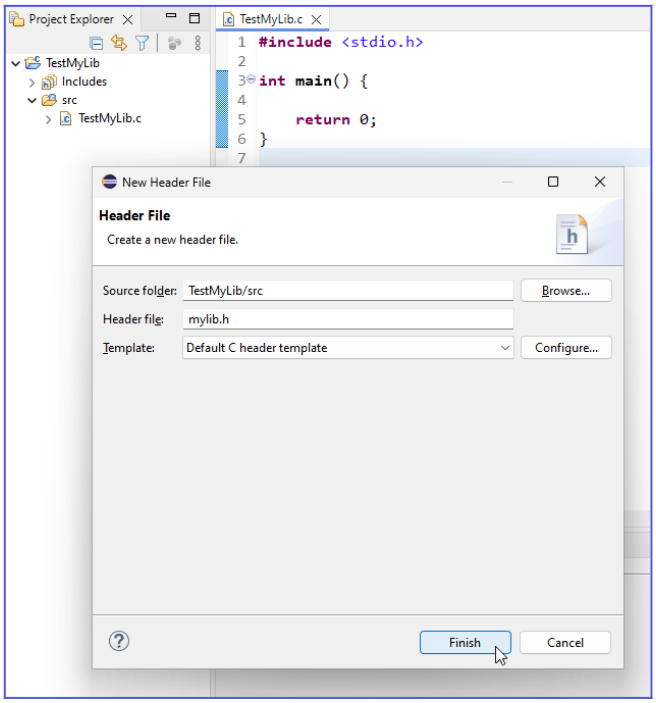


Рис. 5.1. Діалогове вікно створення нового файлу заголовку

Після створення С-проєкту *TestMyLib* в Eclipse IDE, оберіть пункт меню *File \ New \ Header File* та в діалоговому вікні вкажіть ім'я файлу заголовку – *mylib.h* (рис. 5.1), натисніть на кнопку *Finish*. Після створення файлу, оберіть пункт меню *File \ New \ Source File* та в діалоговому вікні вкажіть ім'я файлу *mylib.c*, натисніть на кнопку *Finish*.

Після створення файлів проєкту, перейдіть до кодування. Результат показаний на рис. 5.2.

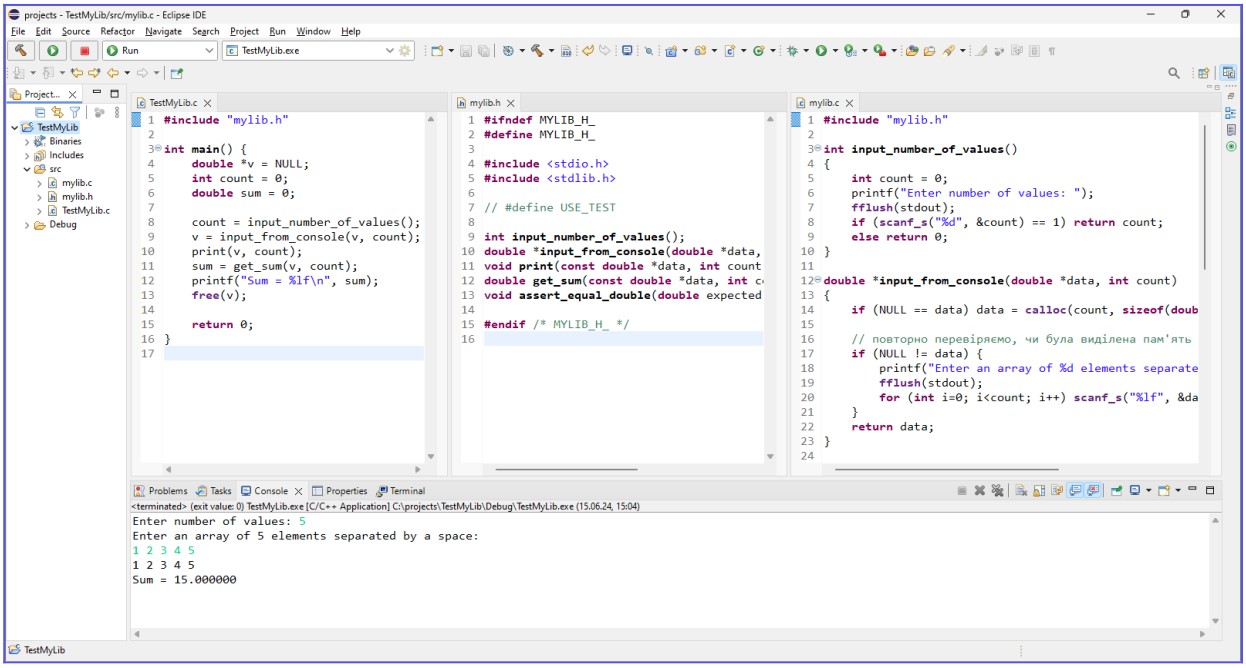


Рис. 5.2. Приклад складу проєкту *TestMyLib*

Таким чином, створений проєкт з двох С-модулів та файлу заголовку.

Завдання

1. Розробити функції обробки динамічних векторів та матриць:

```
// повертають мінімальне значення з вектору або матриці
double get_min_from_vector(const double *data, int count);
double get_min_from_matrix(const double **data, int rows, int cols);

// повертають максимальне значення з вектору або матриці
double get_max_from_vector(const double *data, int count);
double get_max_from_matrix(const double **data, int rows, int cols);

// повертають середнє значення з вектору або матриці
double get_avg_from_vector(const double *data, int count);
double get_avg_from_matrix(const double **data, int rows, int cols);
```

```
// повертають статистику із залученням створених раніше функцій
void get_stat_from_vector(const double *data, int count,
                          double *min, double *max, double *avg);
void get_stat_from_matrix(const double **data, int rows, int cols,
                          double *min, double *max, double *avg);

// введення з консолі елементів вектору або матриці
double *input_vector_from_console(double *data, int count);
double **input_matrix_from_console(double **data, int rows, int cols);

// виведення елементів вектору або матриці на консоль
void print_vector(const double *data, int count);
void print_matrix(const double **data, int rows, int cols);
```

Функції оголосити в файлі *mylib.h*, визначення функцій виконати в модулі *mylib.c*.

2. Розробити програму *TestMyLib* з головним модулем *TestMyLib.c*, що здійснює виклик розроблених функцій, та тестування їх роботи. У програмі повинні бути використані оголошення покажчиків, що представляють в подальшому динамічні вектор та матрицю:

```
double *v;
double **m;
```

Програма повинна запитувати з консолі кількість елементів у векторі та матриці, виділяти пам’ять під елементи цих масивів даних, а також тестувати роботоздатність всіх функцій, що оголошені в *mylib.h*.

Після завершення тестування, в основній програмі повинна бути звільнена пам’ять, яка раніше була виділена під динамічні вектор та матрицю.

3. Запропонувати частину коду для тестування із залученням макросу `USE_TEST`, а також тестові набори вхідних значень та еталонні константи для порівняння.

У звіт до роботи включити код розробленої програми з коментарями, результати її роботи, а також результати тестування функцій. Правила оформлення звіту подані в додатку А.

При захисті роботи володіти знаннями для відповіді на питання викладача за матеріалами лекцій та теоретично-практичної частини роботи.

Робота №6.
Робота з файлами

Мета роботи: закріпити знання та набути навичок роботи з функціями файлового введення/виведення верхнього рівня, а також закріпити навички опрацювання динамічних масивів даних.

Теоретично-практична частина

При роботі з функціями введення/виведення верхнього рівня, використовується спеціальна структура `FILE`. Нехай потрібно виконати збереження вектору значень цілого типу в текстовий файл. Після цієї операції потрібно виконати завантаження даних із збереженого раніше файлу, а потім вивести ці значення на консоль для перевірки. Винесемо операції введення/виведення в окремі функції:

`void save_data_to_text_stream()` – запис вектору цілих в потік, що пов'язаний з файлом, який відкритий в текстовому режимі на запис;
`int *load_data_from_text_stream()` – завантаження вектору з потоку, що пов'язаний з файлом, який відкритий в текстовому режимі на читання.

Для відкриття потоку та його зв'язування з файлом використовують виклик функції `fopen()/fopen_s()`. Закривають потік (окрім стандартних) та скидають всі буфери за допомогою функції `fclose()`.

Примітка 6.1: починаючи зі стандарту C11, описані розширення стандартної бібліотеки, які надають доступ до API *Bounds Checking Interfaces* (описаний у додатку K стандарту мови C). До них має відношення множина *_s* – функцій (*secure function*), в яких проводиться додаткова перевірка на контроль меж пам'яті. Це корисні функції, але це розширення не входить до певної множини бібліотек, які реалізують стандарт. Під платформу MS Windows є реалізації з певними відхиленнями у межах бібліотеки Microsoft CRT, а також в реалізації проєкту MinGW-w64. Для GNU/Linux-сумісних ОС (наприклад, Debian, Ubuntu, Fedora та ін.) та Unix-подібних ОС (наприклад, Apple macOS, FreeBSD та ін.) множина функцій з цього розширення станом на 2025 рік у складі бібліотек – ***відсутня***. Прикладами функцій цього API є: `fopen_s()`, `fscanf_s()`, `sscanf_s()`, `strcpy_s()`, `strcat_s()`, `strnlen_s()` та інші (певні представлені в додатку Б).

Таким чином, маємо наступний фрагмент коду відкриття потоку для запису даних в текстовий файл з використанням функцій стандарту C99:

```
. . .
#define DATA_TXT_FILE   "data.txt"
. . .
FILE *fout = NULL;
int data[] = {1, 2, 3, 4, 5};
. . .
if ((fout = fopen(DATA_TXT_FILE, "wt")) != NULL)
{
      . . .
      // робота з даними та файловим потоком
      . . .
      fclose(fout);
}
. . .
```

При використанні *secure function*, що входять до розширення бібліотеки починаючи зі стандарту C11, маємо наступний код (для бібліотек Microsoft CRT або MinGW-w64 GCC):

```
. . .
#define DATA_TXT_FILE   "data.txt"
. . .
FILE *fout = NULL;
int data[] = {1, 2, 3, 4, 5};
. . .
if (fopen_s(&fout, DATA_TXT_FILE, "wt")) != 0)
{
      . . .
      // робота з даними та файловим потоком
      . . .
      fclose(fout);
}
. . .
```

Примітка 6.2: при використанні середовища розробки MS Visual Studio та стандартної бібліотеки C від Microsoft, для використання класичних C99-функцій потрібно використовувати перед початком коду макрос `_CRT_SECURE_NO_WARNINGS`.

При запису вектору в файл, першим можна записувати кількість елементів (хоча існують і інші підходи). Тому при читанні даних це теж потрібно враховувати.

Приклад реалізації функцій запису та читання вектору значень цілого типу представлений на рис. 6.1.

Зверніть увагу, що в якості роздільника між значеннями у файл записується символ пробілу. Для завантаження значень з потоку використана функція форматованого буферизованого введення `fscanf()` (при реалізації з

використанням *secure function* її виклик потрібно змінити на `fscanf_s()`). Ця функція, як і аналогічна `scanf()`, зчитує данні до пробільного символу або до символу переведення на новий рядок. Повний приклад коду представлений на рис. 6.2.

```
51
52 void save_data_to_text_stream(FILE *stream, const int *data, int count) {
53     if (NULL != stream) { // якщо потік відкритий, тоді
54         fprintf(stream, "%d ", count); // записуємо кількість елементів вектору
55         // записуємо кожний елемент масиву в потік
56         for (int i = 0; i < count; i++) fprintf(stream, "%d ", data[i]);
57         // завершуємо запис в потік переведенням курсору на новий рядок
58         fprintf(stream, "\n");
59     }
60 }
61
62 int* load_data_from_text_stream(FILE *stream, int *data, int *count) {
63     if (NULL != stream) { // якщо потік відкритий, тоді
64         fscanf(stream, "%d", count); // зчитуємо загальну кількіть елементів у векторі
65         // зчитуємо значення кожного елементу вектора
66         for (int i = 0; i < *count; i++) fscanf(stream, "%d", &data[i]);
67     }
68     // повертаємо адресу початку вектора в пам'яті
69     return data;
70 }
```

Рис. 6.1. Приклад реалізації функцій файлового введення/виведення вектору значень цілого типу на базі класичних функцій (C99)

Для тестування оголошені два одновимірні масиви-вектори (рядки 20, 22 – рис. 6.2). Таким чином, тестуємо універсальність роботи функцій як у випадку звичайного масиву *data1*, так й у випадку динамічного одновимірного масиву *data2*.

Перш ніж завантажувати дані з файлу в *data1*, його попередні дані знищуються та знищується інформація про кількість даних (рядки 31, 32).

При переміщенні в потоці (коли відбувається читання), внутрішній покажчик по файлу буде переведений у кінець потоку. Для повторного читання значень в *data2*, його потрібно перевести на початок потоку. Для цього використовується функція `rewind()` (рядок 37).

Після виконання процедури завантаження даних, значення виводяться на консоль (рядки 43 та 45).

Оскільки в програмі маємо динамічний масив даних, то потрібно повернути системі раніше надану пам’ять (рядок 48).

Коли відбувається відкриття файлу в бінарному режимі, то у функціях `fopen()`/`fopen_s()` використовують для параметра *mode* рядок `"rb"` або `"wb"`. Повний список різновидів режимів відкриття файлів приведений в документації, наприклад за посиланням: https://en.cppreference.com/w/c/io/fopen.

Для читання та запису даних в бінарному форматі використовують відповідно функції: `fread()` та `fwrite()` (додаток Б, таблиця Б.1).

```c
// Цей макрос використовують
// для компіляції коду в MS Visual Studio
// без використання secure function
#define _CRT_SECURE_NO_WARNINGS

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

#define MAX_SIZE        5
#define DATA_TXT_FILE   "data.txt"

void save_data_to_text_stream(FILE *stream, const int *data, int count);
int* load_data_from_text_stream(FILE *stream, int *data, int *count);

int main() {
    FILE *fin = NULL;
    FILE *fout = NULL;
    int count = MAX_SIZE;
    // звичайний вектор
    int data1[] = { 1, 2, 3, 4, 5 };
    // динамічний вектор
    int *data2 = (int*) malloc(sizeof(int) * MAX_SIZE);

    // записуємо значення вектору в тектовий файл
    if ((fout = fopen(DATA_TXT_FILE, "wt")) != NULL) {
        save_data_to_text_stream(fout, data1, count);
        fclose(fout);
    }

    // очищуємо вектор перед завантаженням даних
    for (int i = 0; i < count; i++) data1[i] = 0;
    count = 0;

    // завантажуємо дані в обидві вектори
    if ((fin = fopen(DATA_TXT_FILE, "rt")) != NULL) {
        load_data_from_text_stream(fin, data1, &count);
        rewind(fin); // переміщуємо внутрішній покажчик позиції потоку на початок
        data2 = load_data_from_text_stream(fin, data2, &count);
        fclose(fin);
    }

    // виведення значень елементів векторів на консоль
    for (int i = 0; i < count; i++) printf("%d ", data1[i]);
    printf("\n");
    for (int i = 0; i < count; i++) printf("%d ", data2[i]);
    printf("\n");

    free(data2);
    return 0;
}
```

Рис. 6.2. Приклад коду функції `main()` з викликом функцій введення/виведення вектору значень цілого типу на базі класичних функцій (C99)

Завдання

1. В контексті проєкту *TestMyLib*, реалізувати код нових функцій та модифікувати коди розроблених раніше в роботі №5 функцій для підтримки файлового введення/виведення:

```
double *load_vector_from_text_stream(FILE *stream, double *data, int count);
double **load_matrix_from_text_stream(FILE *stream, double **data, int rows, int cols);
double *load_vector_from_binary_stream(FILE *stream, double *data, int count);
double **load_matrix_from_binary_stream(FILE *stream, double **data, int rows, int cols);

void save_vector_to_text_stream(FILE *stream, const double *data, int count);
void save_matrix_to_text_stream(FILE *stream, const double **data, int rows, int cols);
void save_vector_to_binary_stream(FILE *stream, const double *data, int count);
void save_matrix_to_binary_stream(FILE *stream, const double **data, int rows, int cols);

void print_vector(FILE *stream, const double *data, int count);
void print_matrix(FILE *stream, const double **data, int rows, int cols);
```

При цьому використовувати такі формати зберігання вектора та матриці в бінарному файлі:

– у випадку вектору даних:

<кількість_елементів_вектора> <елемент_1> … <елемент_n>

– у випадку матриці даних:

<кількість_рядків> <кількість_стовпців> <елемент_1> … <елемент_n>

2. Протестувати в основній програмі виклики нових функцій.

Всі зміни вносити в файли *mylib.h*, *mylib.c*, *TestMyLib.c*, що були розроблені під час виконання роботи №5.

У звіт до роботи включити код модифікованої програми *TestMyLib* з коментарями, результати її роботи, результати тестування функцій, вміст файлів даних. Правила оформлення звіту подані в додатку А.

При захисті роботи володіти знаннями для відповіді на питання викладача за матеріалами лекцій та теоретично-практичної частини роботи.

Самостійна робота

Мета роботи: показати знання та навички роботи з простими динамічними структурами даних.

Завдання

В контексті проєкту *LinkedList*, реалізувати функції для роботи з односпрямованим динамічним списком певних структур.

Проєкт повинен включати три файли: *linked_list.c*, *linked_list.h* та *main.c*.

linked_list.h

```c
#ifndef LINKED_LIST_H_
#define LINKED_LIST_H_

typedef struct _tagNode NODE, *PNODE;

struct _tagNode {
    int data;
    PNODE next;
};

PNODE add_element_to_list(PNODE head, int data);
int get_size_list(const PNODE head);
const PNODE get_element_from_list(const PNODE head, int index);
PNODE delete_element_from_list(PNODE head, int index);
PNODE clear_list(PNODE head);
void print_list(const PNODE head);

#endif

```

main.c

```c
#include "linked_list.h"
#include <stdio.h>

#define MAX 10

int main(int argc, char *argv[])
{
    PNODE p_head = NULL;

    for (int i=1; i<=MAX; i++) p_head = add_element_to_list(p_head, i);
    print_list(p_head);
    int count = get_size_list(p_head);
    printf("Count: %d\n", count);
    const PNODE pa = get_element_from_list(p_head, 5);
    printf("\nValue on 5 index: %d\n\n", pa->data);

    p_head = delete_element_from_list(p_head, 0);
    print_list(p_head);
    count = get_size_list(p_head);
    printf("Count after delete on 0 index: %d\n", count);

    p_head = delete_element_from_list(p_head, 8);
    print_list(p_head);
    count = get_size_list(p_head);
    printf("Count after delete on 8 index: %d\n", count);

    p_head = delete_element_from_list(p_head, 4);
    print_list(p_head);
    count = get_size_list(p_head);
    printf("Count after delete on 4 index: %d\n", count);

    p_head = clear_list(p_head);
    count = get_size_list(p_head);
    printf("\nCount after clear: %d\n", count);

    return 0;
}

```

Рис. 7.1. Вміст файлів *linked_list.h* та *main.c*

```
1 2 3 4 5 6 7 8 9 10
Count: 10

Value on 5 index: 6

2 3 4 5 6 7 8 9 10
Count after delete on 0 index: 9
2 3 4 5 6 7 8 9
Count after delete on 8 index: 8
2 3 4 5 7 8 9
Count after delete on 4 index: 7

Count after clear: 0
```

Рис. 7.2. Результат виконання програми *LinkedList*

На рис. 7.1 представлений вміст двох файлів. Їх реалізацію змінювати не можна! Потрібно виконати реалізацію запропонованих функцій у модулі *linked_list.c*.

На рис. 7.2 представлений результат виконання програми.

У звіт до виконаного завдання включити код трьох файлів проєкту з коментарями та результати роботи програми.

Правила оформлення звіту подані в додатку А.

При захисті завдання володіти знаннями для відповіді на питання викладача за матеріалами лекцій та питань по коду проєкту.

Критерії оцінювання виконання робіт

Виконання завдань, які описані в практичних частинах робіт, та захист результатів, що подані здобувачем у звітах, є обов'язковим. При наявності в роботі декількох варіантів завдання, конкретний варіант для виконання призначає викладач.

В залежності від складності практичної частини роботи, на виконання завдань відводиться певна кількість годин, обсягом, що представлений в робочій програмі дисципліни.

Після виконання практичної частини роботи, здобувач складає звіт, в якому повинен описати хід виконання завдання та надати висновки щодо отриманих результатів роботи. Вимоги по оформленню звітності подані в додатку А.

Захист звіту з виконаної роботи проводиться під час проведення заняття у встановлений розкладом основний час або у додатковий час, який погоджений із викладачем.

Узагальнені критерії оцінки виконаної роботи наведені у таблиці:

Критерії оцінювання виконання та захисту роботи здобувачем	*Рейтингова оцінка*	*Інституційна оцінка*
Результати виконання роботи вірні згідно вимог, описаних в завданні та наданих викладачем в ході лекційних занять. Звіт оформлений згідно вимог, описаних в додатку А. На всі поставлені питання викладача при захисті роботи надані вірні відповіді. Робота захищена.	90 – 100	відмінно
Результати виконання роботи вірні згідно вимог, описаних в завданні та наданих викладачем в ході лекційних занять. Звіт оформлений згідно вимог, описаних в додатку А. На 80% поставлених питань викладача при захисті роботи були надані вірні відповіді. Робота захищена.	80 – 89	добре
Результати виконання роботи вірні згідно вимог, описаних в завданні та наданих викладачем в ході лекційних занять. Звіт оформлений згідно вимог, описаних в додатку А. На 70% поставлених питань викладача при захисті роботи були надані вірні відповіді. Робота захищена.	74 – 79	
Результати виконання роботи містять помилки. Є відхилення від поставлених вимог оформлення звітності. Під час захисту здобувач надав 50% вірних відповідей на поставлені питання викладача. Робота захищена.	60 – 73	задовільно
Результати виконання роботи невірні або звіт відсутній, або здобувач не може під час захисту результатів роботи відповісти на питання викладача. Робота не була захищена.	0 – 59	незадовільно

Перелік рекомендованих джерел

1. ДСТУ ISO 5807:2016 (ISO 5807:1985, IDT). Оброблення інформації. Символи та угоди щодо документації стосовно даних, програм та системних блок-схем, схем мережевих програм та схем системних ресурсів. Київ : ДП «УкрНДНЦ», 2016.

2. International standard ISO/IEC 9899:2024 “Information technology – Programming languages – C”. ISO, 2024. 759 p. URL: https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3220.pdf. (дата звернення: 24.03.2025).

3. The GNU C Library Reference Manual. Free Software Foundation, Inc., 2023. 239 p. URL: https://www.gnu.org/software/libc/manual/pdf/libc.pdf. (дата звернення: 24.03.2025).

4. Гаркуша І.М. Конспект лекцій з дисципліни “Програмування”. Для здобувачів галузі знань 12 “Інформаційні технології” спеціальностей 123 “Комп’ютерна інженерія” та 126 “Інформаційні системи та технології”. 3-є вид., перероб. та доп. Дніпро : НТУ «ДП», 2023. 102 с.

5. Карпенко Н.В., Герасимов В.В. Сучасний підхід до програмування на мові С від нульового до просунутого рівня : навч. посіб. Дніпро : Ліра, 2022. 418 с.

6. Карпенко Н.В. Розробка програм на мові С у сучасних середовищах : навч-метод. посіб. Дніпро : ЛІРА, 2016. 144 с.

7. Програмування та алгоритмічні мови 1. Алгоритмізація та основи програмування: Конспект лекцій : навч. посіб. для студ. спеціальності 124 «Системний аналіз», освітньо-професійні програми «Системний аналіз та управління», «Системний аналіз фінансового ринку» / КПІ ім. Ігоря Сікорського; уклад.: І.В. Назарчук. Електронні текстові дані (1 файл: 1,1 Мбайт). Київ : КПІ ім. Ігоря Сікорського, 2020. 140 с.

Додаток А.
Вимоги до оформлення звітності

Звіт оформлюється на листах формату А4 та включає титульний аркуш (зразок оформлення представлений на рис. А.1) та опис роботи за представленим нижче зразком.

У разі захисту звіту з виконання роботи при аудиторній формі навчання, звіт друкується та надається викладачу на розгляд.

Для захисту виконаної роботи у дистанційній (online) формі навчання, звіт подається до захисту в електронному форматі PDF (Portable Document Format) та надсилається викладачеві через засоби online-спілкування для розгляду.

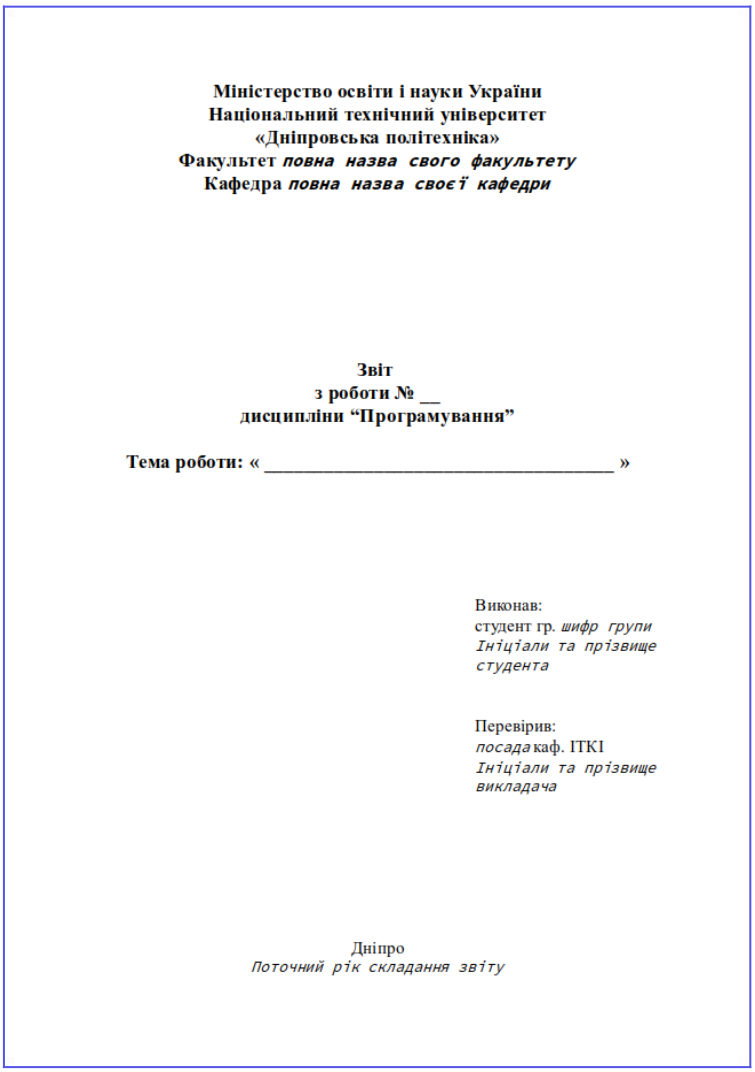
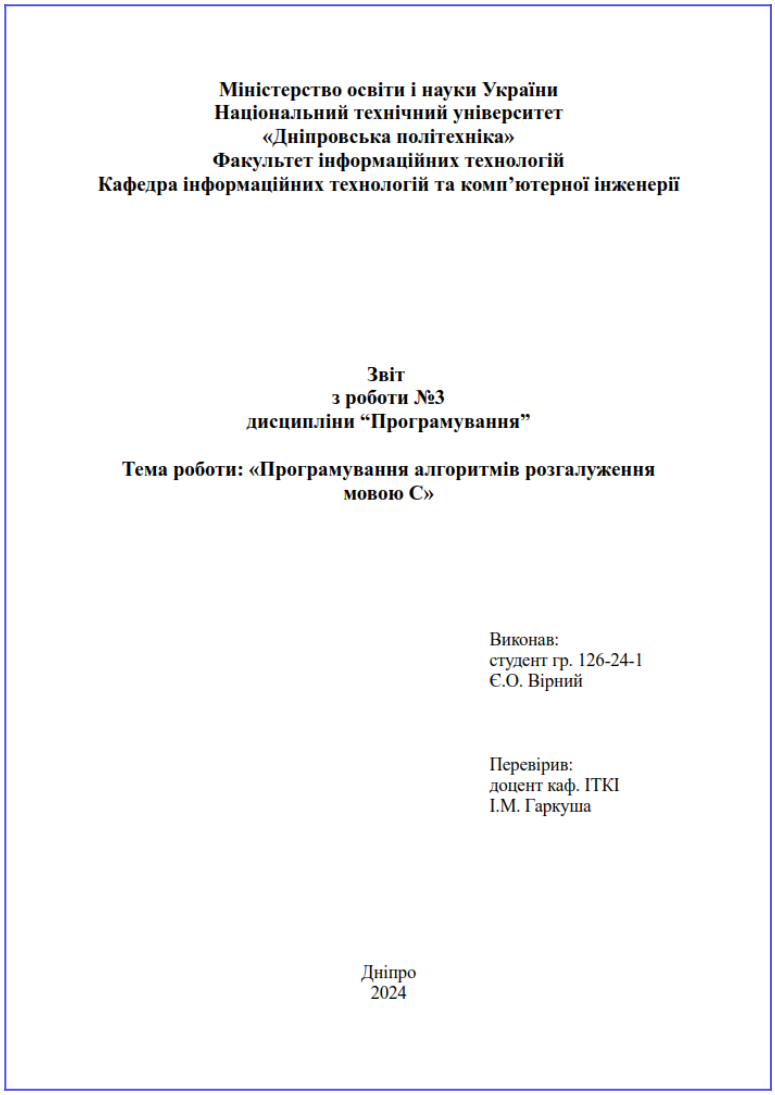


Рис. А.1. Форма титульного аркуша звіту



Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»
Факультет інформаційних технологій
Кафедра інформаційних технологій та комп'ютерної інженерії

Звіт
з роботи №3
дисципліни "Програмування"

Тема роботи: «Програмування алгоритмів розгалуження мовою С»

Виконав:
студент гр. 126-24-1
Є.О. Вірний

Перевірив:
доцент каф. ІТКІ
І.М. Гаркуша

Дніпро
2024

Рис. А.2. Приклад оформленого титульного аркуша звіту

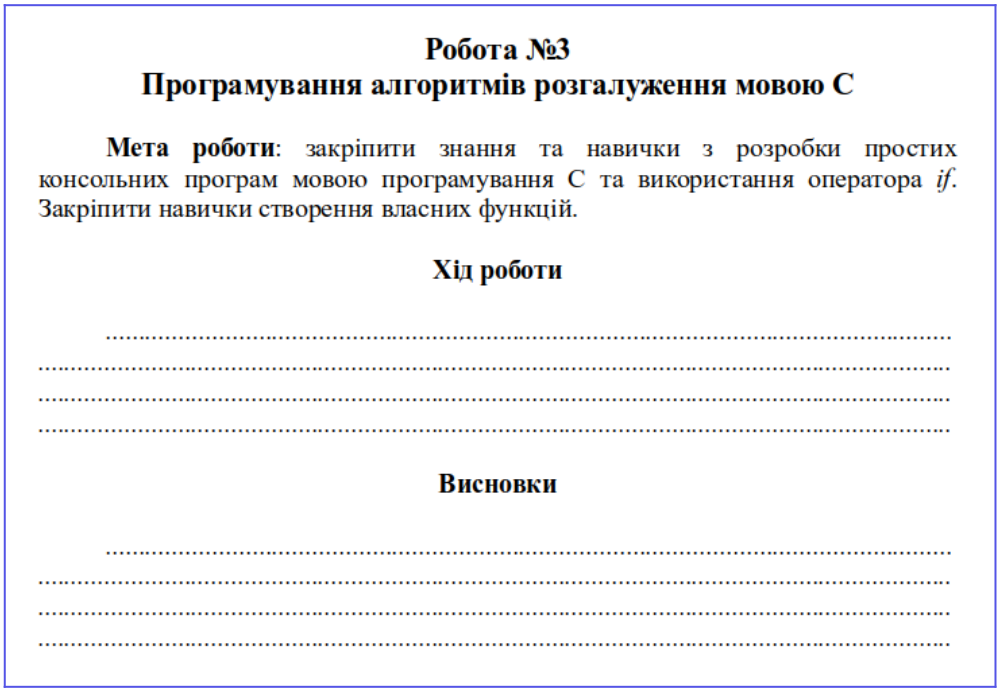
Опис виконаної роботи в звіті розпочинається з наступного аркуша. В горі сторінки, по центру, вказується номер роботи та її тема. Після цього вказується мета роботи. Через рядок по центру сторінки йдуть слова *Хід роботи* і далі через рядок розпочинається основний текст опису виконаної роботи.

Після завершення опису роботи, через рядок, йде частина, яка присвячена висновкам по проведеній роботі. Вона починається зі слова *Висновки*, розміщеного по центру листа та через рядок починається текст з висновками по роботі.

Приклад загального вмісту опису роботи представлений на рис. А.3.

Написання основного тексту ходу роботи та висновків йде **від третьої особи**!!! Тобто, є невірним використання у звіті висловів, на кшталт: "В ході роботи я розробив алгоритм розгалуження". Замість такого вислову вірно буде написати: "В ході роботи розроблений алгоритм розгалуження". Або замість вислову, на кшталт: "Під час виконання роботи ми

опанували використання оператору розгалуження”, вірно буде написати: “Під час виконання роботи опановано використання оператору розгалуження”.



Робота №3
Програмування алгоритмів розгалуження мовою С

Мета роботи: закріпити знання та навички з розробки простих консольних програм мовою програмування С та використання оператора *if*. Закріпити навички створення власних функцій.

Хід роботи

.....
.....
.....
.....

Висновки

.....
.....
.....
.....

Рис. А.3. Приклад загального подання опису роботи

Текст звіту оформляється шрифтом Times New Roman з розміром 14, міжрядковий інтервал 1 – 1,5. Абзацний відступ – 5 символів.

Шрифтом з розміром 10 оформлюють тексти кодів програм. При цьому бажано використовувати один з наступних моноширинних шрифтів: Source Code Pro, Monospace, Consolas, DejaVu Sans Mono, Courier New, Monospaced, Menlo, SF Mono або подібні.

Вирівнювання тексту опису в звіті – по ширині сторінки.

Вирівнювання тексту коду програм в звіті – ліворуч.

Вирівнювання рисунків та підписів до них – по центру сторінки.

Вирівнювання таблиць (якщо такі будуть) – по центру сторінки.

Якщо код програми не є великим за обсягом, то його можна подати у вигляді знімку з екрану, причому фон повинен бути білого кольору, а текст коду програми повинен добре виділятися. Приклад такого знімку з екрану представлений на рис. А.4. Такий рисунок потрібно розмістити по центру сторінки звіту та надати по центру підпис для нього. Наприклад:

Рисунок 1 – Код програми за пунктом 1 завдання роботи №3

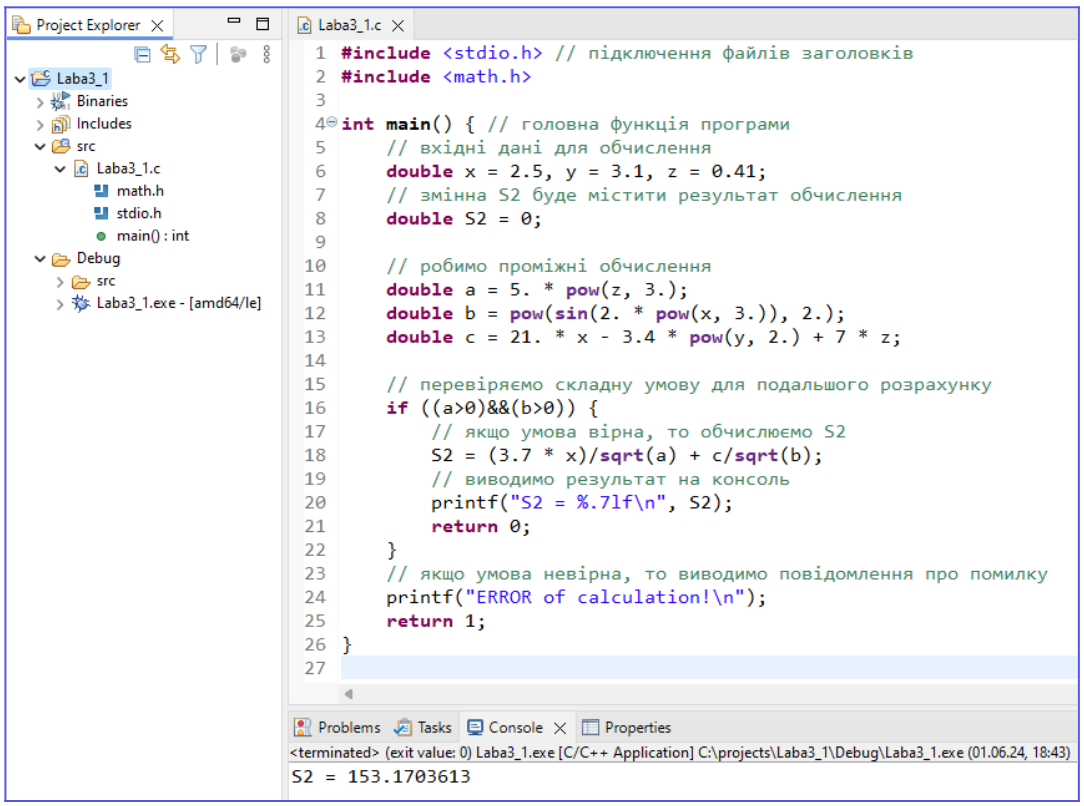


Рис. А.4. Приклад знімка з екрану коду програми для звіту

Якщо в тексті звіту наводиться рисунок, то на нього обов'язково повинно бути посилання з тексту опису ходу роботи та пояснення, щодо певних елементів коду, які представлені на вказаному рисунку. Наприклад, потрібно, вказуючи номер рядку, прокоментувати певну використану функцію та коротко описати її особливості.

В роботах є завдання, які містять варіанти. При отриманні варіанту виконання від викладача, цей номер варіанту повинен бути відображений на початку опису ходу роботи. Якщо варіант завдання містить математичний вираз для підрахунку, то такий вираз також повинен бути присутнім у звіті. Для створення математичного виразу потрібно скористатися певними інструментами. Математичний вираз розміщують по центру сторінки та, при необхідності, нумерують. Наприклад:

$$S2=\frac{3{,}7\,x}{\sqrt{5\,z^3}}+\frac{21\,x-3{,}4\,y^2+7\,z}{\sqrt{\sin^2 2\,x^3}} \qquad (1)$$

При складанні тексту висновків, потрібно вказати на результати щодо досягнення поставленої мети роботи. Вказати особливості виконання роботи та складнощі. При цьому доречно залучати вислови на кшталт: "В роботі розглянуті...", "...виконані...", "Отриманий результат дозволив...", "Використання умов дозволило..." та ін.

Додаток Б.
Деякі функції стандартної бібліотеки мови С

Стандартна бібліотека мови С (*C Standard Library* або *libc*) складається з файлів заголовків (header-файлів) та файлів бібліотек (lib- та/або a-файлів). Перелік цих файлів та їх вміст залежить від певної редакції стандарту мови, певних розширень цього стандарту, певних програмних платформ та компіляторів, до складу яких входить бібліотека.

Найвідомішими реалізаціями з власними розширеннями бібліотеки є:

– C POSIX Library (у складі стандарту POSIX);

– GNU C Library (glibc, у складі компіляторів з проєктів GNU та LLVM);

– Microsoft CRT (C RunTime), Microsoft UCRT (Universal C RunTime).

Повний перелік файлів заголовків стандартної бібліотеки мови С можна подивитися за посиланням:

https://en.cppreference.com/w/c/header

Почитати додатково про бібліотеки та їх функції у складі GNU, Microsoft та POSIX (станом на початок 2025 року) можна за відповідними посиланнями:

https://www.gnu.org/software/libc/manual/pdf/libc.pdf

https://learn.microsoft.com/en-us/cpp/c-runtime-library

https://pubs.opengroup.org/onlinepubs/9699919799/

В цьому додатку перелічені функції, макроси та макроконстанти бібліотеки, які найбільш часто використовуються в задачах та роботах до поточного курсу. Тобто в таблицях нижче перелічені найвідоміші основні функції бібліотеки і це означає, що коло функцій у наведених файлах заголовків значно більше ніж наведено в цьому додатку в таблицях.

Таблиця Б.1

Бібліотечні функції введення/виведення верхнього рівня
(файл заголовку stdio.h)

Ім'я функції	*Загальний опис*
`printf`	Форматоване виведення даних в стандартний потік stdout https://en.cppreference.com/w/c/io/fprintf
`scanf`, `scanf_s`	Форматоване введення даних з потоку stdin https://en.cppreference.com/w/c/io/fscanf
`puts`	Виведення в потік stdout С-рядку з переведенням курсору на новий рядок https://en.cppreference.com/w/c/io/puts
`gets_s`	Введення символьного С-рядку з потоку stdin з визначеним розміром

	https://en.cppreference.com/w/c/io/gets (колишня функція `gets()` є застарілою та була вилучена в стандарті C11)
`putc`, `fputc`	Виведення символу у вказаний потік https://en.cppreference.com/w/c/io/fputc
`putchar`	Виведення символу у потік stdout https://en.cppreference.com/w/c/io/putchar
`getc`, `fgetc`	Читання символу із вказаного потоку https://en.cppreference.com/w/c/io/fgetc
`getchar`	Читання символу із потоку stdin https://en.cppreference.com/w/c/io/getchar
`spintf`	Форматований запис даних в С-рядок https://en.cppreference.com/w/c/io/fprintf
`sscanf`, `sscanf_s`	Зчитує форматовані дані з С-рядку https://en.cppreference.com/w/c/io/fscanf
`fopen`	Відкриває файл для певної операції та зв'язує його з потоком https://en.cppreference.com/w/c/io/fopen
`freopen`	Відкриває файл для певної операції та зв'язує його із вже існуючим потоком (повторне відкриття потоку в новому режимі) https://en.cppreference.com/w/c/io/freopen
`fclose`	Закриття певного файлового потоку окрім стандартного https://en.cppreference.com/w/c/io/fclose
`feof`	Перевіряє, чи не був досягнений кінець файлового потоку https://en.cppreference.com/w/c/io/feof
`rewind`	Переміщує внутрішній покажчик по файлу на початок файлового потоку https://en.cppreference.com/w/c/io/rewind
`fseek`	Переміщує внутрішній покажчик по файлу, який є зв'язаним з певним потоком, в задану позицію https://en.cppreference.com/w/c/io/fseek
`ftell`	Отримання поточної позиції внутрішнього покажчика у файлі, зв'язаного з певним потоком https://en.cppreference.com/w/c/io/ftell
`fprintf`	Форматоване виведення даних у певний файловий потік https://en.cppreference.com/w/c/io/fprintf
`fscanf`, `fscanf_s`	Форматоване введення даних з певного файлового потоку https://en.cppreference.com/w/c/io/fscanf
`fputs`	Виведення в певний файловий потік С-рядку https://en.cppreference.com/w/c/io/fputs
`fgets`	Введення символьного С-рядку з певного файлового потоку з визначеною кількістю символів https://en.cppreference.com/w/c/io/fgets
`fread`	Неформатоване читання даних визначеного об'єму в буфер з певного файлового потоку https://en.cppreference.com/w/c/io/fread
`fwrite`	Неформатований запис даних визначеного об'єму з буфера у певний файловий потік https://en.cppreference.com/w/c/io/fwrite
`fflush`	Запис незбереженого буфера вказаного файлового потоку на пов'язаний з ним зовнішній пристрій https://en.cppreference.com/w/c/io/fflush
`tmpfile`,	Відкриває файловий потік, що пов'язаний з новим тимчасовим файлом

`tmpfile_s`	https://en.cppreference.com/w/c/io/tmpfile
`tmpnam`, `tmpnam_s`	Генерує ім’я тимчасового файлу для розміщення у вказаному каталозі https://en.cppreference.com/w/c/io/tmpnam

В *stdio.h* оголошені певні макроконстанти (наприклад: EOF, FOPEN_MAX, FILENAME_MAX, BUFSIZ, SEEK_SET, SEEK_CUR, SEEK_END, TMP_MAX, TMP_MAX_S), а також тип даних FILE (структура, яка містить відомості про потік) для роботи з операціями введення/виведення верхнього рівня та глобальні змінні для роботи зі стандартними потоками: `stdin`, `stdout`, `stderr` (https://en.cppreference.com/w/c/io).

Таблиця Б.2

Бібліотечні функції загального призначення (файл заголовку stdlib.h)

Ім’я функції	*Загальний опис*
`atoi`, `atol`, `atoll`	Перетворення послідовності символів С-рядку в число типів int, long та long long https://en.cppreference.com/w/c/string/byte/atoi
`atof`	Перетворення послідовності символів С-рядку в число типу double https://en.cppreference.com/w/c/string/byte/atof
`strtol`, `strtoll`	Перетворення послідовності символів С-рядку в число типів long та long long https://en.cppreference.com/w/c/string/byte/strtol
`strtoul`, `strtoull`	Перетворення послідовності символів С-рядку в число типу unsigned long та unsigned long long https://en.cppreference.com/w/c/string/byte/strtoul
`strtof`, `strtod`, `strtold`	Перетворення послідовності символів С-рядку в число типів float, double та long double https://en.cppreference.com/w/c/string/byte/strtof
`system`	Викликає командний процесор ОС та передає йому команду на виконання https://en.cppreference.com/w/c/program/system
`exit`	Викликає завершення програми з вказівкою коду завершення та з очищенням ресурсів програми https://en.cppreference.com/w/c/program/exit
`abort`	Викликає аварійне завершення програми без очищення ресурсів програми https://en.cppreference.com/w/c/program/abort
`getenv`, `getenv_s`	Надає доступ до списку системних змінних ОС https://en.cppreference.com/w/c/program/getenv
`malloc`	Виконує динамічний розподіл пам’яті https://en.cppreference.com/w/c/memory/malloc
`calloc`	Виконує динамічний розподіл пам’яті та обнуляє її https://en.cppreference.com/w/c/memory/calloc
`realloc`	Розширює або звужує раніше динамічно виділену пам’ять https://en.cppreference.com/w/c/memory/realloc
`free`	Звільняє раніше динамічно виділену пам’ять https://en.cppreference.com/w/c/memory/free
`qsort`, `qsort_s`	Сортує елементи невизначеного типу https://en.cppreference.com/w/c/algorithm/qsort

`bsearch`, `bsearch_s`	Пошук в масиві елементу невизначеного типу https://en.cppreference.com/w/c/algorithm/bsearch
`abs`, `labs`, `llabs`	Повернення абсолютного значення аргументу цілого типу https://en.cppreference.com/w/c/numeric/math/abs
`srand`	Готує генератор псевдовипадкових чисел для подальшого використання функцією `rand()` https://en.cppreference.com/w/c/numeric/random/srand
`rand`	Повертає псевдовипадкове ціле значення у межах [0; RAND_MAX] https://en.cppreference.com/w/c/numeric/random/rand

В *stdlib.h* також оголошені певні макроконстанти (наприклад: EXIT_SUCCESS, EXIT_FAILURE, RAND_MAX).

В *stdlib.h* можуть бути присутні застарілі функції, на кшталт `ecvt()`, `fcvt()`, `gcvt()`. За вимогами стандарту POSIX.1-2008 їх специфікації були виключені та замість них рекомендується використовувати функції, на кшталт `sprintf()` або `snprintf()`.

В *stdlib.h* можна також зустріти множини функцій `itoa()`, `ltoa()` або подібних, які фактично входять у розширення бібліотеки від Microsoft та не входять до складу бібліотеки за стандартом С.

Таблиця Б.3

Бібліотечні функції для перевірки символів та їх перетворення (файл заголовку ctype.h)

Ім'я функції	*Загальний опис*
`isalnum`	Перевірка на букву або цифру https://en.cppreference.com/w/c/string/byte/isalnum
`isalpha`	Перевірка на букву https://en.cppreference.com/w/c/string/byte/isalpha
`iscntrl`	Перевірка на символ керування https://en.cppreference.com/w/c/string/byte/iscntrl
`isdigit`	Перевірка на цифру десяткової системи числення https://en.cppreference.com/w/c/string/byte/isdigit
`isxdigit`	Перевірка на цифру шістнадцяткової системи числення https://en.cppreference.com/w/c/string/byte/isxdigit
`isgraph`	Перевіряє, чи має певний символ графічне представлення https://en.cppreference.com/w/c/string/byte/isgraph
`islower`	Перевіряє, чи є певний символ у нижньому регістрі https://en.cppreference.com/w/c/string/byte/islower
`isupper`	Перевіряє, чи є певний символ у верхньому регістрі https://en.cppreference.com/w/c/string/byte/isupper
`isprint`	Перевіряє, чи є певний символ символом для друку https://en.cppreference.com/w/c/string/byte/isprint
`ispunct`	Перевіряє, чи є певний символ символом пунктуації https://en.cppreference.com/w/c/string/byte/ispunct
`isspace`	Перевіряє, чи є певний символ пробільним символом https://en.cppreference.com/w/c/string/byte/isspace

`isblank`	Перевіряє, чи є певний символ пробілом або горизонтальною табуляцією https://en.cppreference.com/w/c/string/byte/isblank
`tolower`	Повертає код символу для нижнього регістру https://en.cppreference.com/w/c/string/byte/tolower
`toupper`	Повертає код символу для верхнього регістру https://en.cppreference.com/w/c/string/byte/toupper

В *ctype.h* можуть зустрітися функції `isascii()` та `toascii()`. Стандарт POSIX.1-2008 помічає їх як застарілі та не рекомендує до використання.

Таблиця Б.4

Бібліотечні функції для роботи з С-рядками (файл заголовку string.h)

Ім'я функції	*Загальний опис*
`strcat`, `strcat_s`	Конкатенація (об'єднання) двох рядків https://en.cppreference.com/w/c/string/byte/strcat
`strncat`, `strncat_s`	Додає n-символів до рядку https://en.cppreference.com/w/c/string/byte/strncat
`strchr`	Пошук спочатку рядка першого входження заданого символу https://en.cppreference.com/w/c/string/byte/strchr
`strrchr`	Пошук з кінця рядка першого входження заданого символу https://en.cppreference.com/w/c/string/byte/strrchr
`strstr`	Пошук підрядка в рядку символів https://en.cppreference.com/w/c/string/byte/strstr
`strcmp`	Порівняння двох рядків з врахуванням регістру символів https://en.cppreference.com/w/c/string/byte/strcmp
`strncmp`	Порівняння двох рядків з врахуванням регістру символів та вказаної кількості символів https://en.cppreference.com/w/c/string/byte/strncmp
`strcpy`, `strcpy_s`	Копіювання одного рядку в інший https://en.cppreference.com/w/c/string/byte/strcpy
`strncpy`, `strncpy_s`	Копіювання одного рядку в інший з врахуванням кількості символів https://en.cppreference.com/w/c/string/byte/strncpy
`strdup`	Дублювання рядку (після використання змінної результату, треба використовувати виклик функції `free()`) https://en.cppreference.com/w/c/string/byte/strdup
`strndup`	Дублювання рядку з врахуванням кількості пам'яті (після використання змінної результату, треба використовувати виклик функції `free()`) https://en.cppreference.com/w/c/string/byte/strndup
`strlen`, `strnlen_s`	Повертає довжину рядка https://en.cppreference.com/w/c/string/byte/strlen
`strtok`, `strtok_s`	Знаходить наступний токен у рядку з врахуванням певних параметрів https://en.cppreference.com/w/c/string/byte/strtok

В *string.h* можуть бути знайдені функції, на кшталт: `strlwr()`, `strupr()`, `stricmp()`, `strnicmp()`, `strcmpi()`, `_strset()`, `_strset_s()`, які фактично входять у розширення бібліотеки від Microsoft та не входять до складу бібліотеки за стандартом С.

При використанні стандартної бібліотеки мови С без додаткових розширень (наприклад, розширень Microsoft), для виконання порівняння двох рядків без врахування регістру символів, символи кожного рядку приводять до нижнього регістру. Після чого проводять порівняння однією з функцій: `strcmp()` або `strncmp()`.

Таблиця Б.5

Бібліотечні математичні функції (файл заголовку math.h)

Ім'я функції або макрофункції	*Загальний опис*
`fabs, fabsf, fabsl, fabsd32, fabsd64, fabsd128`	Повернення абсолютного значення аргументу https://en.cppreference.com/w/c/numeric/math/fabs
`fmod, fmodf, fmodl`	Обчислення залишку з плаваючою комою від ділення двох значень https://en.cppreference.com/w/c/numeric/math/fmod
`fmax, fmaxf, fmaxl`	Повернення максимального з двох значень https://en.cppreference.com/w/c/numeric/math/fmax
`fmin, fminf, fminl`	Повернення мінімального з двох значень https://en.cppreference.com/w/c/numeric/math/fmin
`exp, expf, expl`	Повернення розрахунку *e* в заданому ступені (e^x) https://en.cppreference.com/w/c/numeric/math/exp
`log, logf, logl`	Обчислення натурального логарифму певного значення (*ln x*) https://en.cppreference.com/w/c/numeric/math/log
`log10, log10f, log10l`	Обчислення логарифму за основою 10 певного значення ($log_{10}\, x$ або *lg x*) https://en.cppreference.com/w/c/numeric/math/log10
`log2, log2f, log2l`	Обчислення логарифму за основою 2 певного значення ($log_2\, x$ або *lb x*) https://en.cppreference.com/w/c/numeric/math/log2
`pow, powf, powl`	Возведення певного значення в певний ступінь https://en.cppreference.com/w/c/numeric/math/pow
`sqrt, sqrtf, sqrtl`	Повернення результату обчислення кореня квадратного для певного значення https://en.cppreference.com/w/c/numeric/math/sqrt
`cbrt, cbrtf, cbrtl`	Повернення результату обчислення кореня кубічного для певного значення https://en.cppreference.com/w/c/numeric/math/cbrt
`hypot, hypotf, hypotl`	Обчислення довжини гіпотенузи прямокутного трикутника (корінь квадратний із суми квадратів довжин катетів) https://en.cppreference.com/w/c/numeric/math/hypot
`sin, sinf, sinl`	Обчислення синусу (значення аргументу подається в радіанах) https://en.cppreference.com/w/c/numeric/math/sin
`cos, cosf,`	Обчислення косинусу (значення аргументу подається в радіанах)

`cosl`	https://en.cppreference.com/w/c/numeric/math/cos
`tan, tanf, tanl`	Обчислення тангенсу (значення аргументу подається в радіанах) https://en.cppreference.com/w/c/numeric/math/tan
`asin, asinf, asinl`	Обчислення арксинусу (значення аргументу повинно бути в діапазоні [–1; +1]) https://en.cppreference.com/w/c/numeric/math/asin
`acos, acosf, acosl`	Обчислення арккосинусу (значення аргументу повинно бути в діапазоні [–1; +1]) https://en.cppreference.com/w/c/numeric/math/acos
`atan, atanf, atanl`	Обчислення арктангенсу https://en.cppreference.com/w/c/numeric/math/atan
`ceil, ceilf, ceill`	Обчислює найменше ціле значення не менше за аргумент https://en.cppreference.com/w/c/numeric/math/ceil
`floor, floorf, floorl`	Обчислює найбільше ціле значення, яке не перевищує аргумент https://en.cppreference.com/w/c/numeric/math/floor
`trunc, truncf, truncl`	Обчислює найближче ціле значення, величина якого не перевищує аргумент https://en.cppreference.com/w/c/numeric/math/trunc
`round, roundf, roundl, lround, lroundf, lroundl, llround, llroundf, llroundl`	Обчислює найближче ціле значення до значення аргументу в різних типах даних, округляючи наполовину від нуля, незалежно від поточного режиму округлення https://en.cppreference.com/w/c/numeric/math/round
`modf, modff, modfl`	Розкладає задане значення аргументу з плаваючою комою на цілу та дробову частини, кожна з яких має той самий тип та знак, що і аргумент https://en.cppreference.com/w/c/numeric/math/modf
`isnan`	Макрофункція визначає, чи не є числом певне значення https://en.cppreference.com/w/c/numeric/math/isnan
`isinf`	Макрофункція визначає, чи є певне значення позитивною або негативною нескінченністю https://en.cppreference.com/w/c/numeric/math/isinf

В *math.h* оголошені певні макроконстанти (наприклад: NAN, INFINITY та інші). Повний перелік математичних функцій та макроконстант можна подивитися за посиланням: https://en.cppreference.com/w/c/numeric/math.

Стандарт С не регламентує особливих математичних констант на кшталт значень π або *e*, але їх значення, як правило, присутні в *math.h* та залучаються при використанні певного макросу або у певних різновидах бібліотек доступні за замовчуванням. Наприклад, при використанні компіляторів GCC (в GNU/Linux-сумісних ОС) та Clang (зі складу Command Line Tools в Apple macOS) додаткових макросів використовувати не треба (тільки за потреби для підвищення точності значення, наприклад, числа π). При використанні

інструментів розробника від Microsoft для доступу до подібних констант, перед підключенням *math.h*, потрібно залучити макрос `_USE_MATH_DEFINES`.

Таблиця Б.6

Бібліотечні функції для роботи з датою та часом (файл заголовку time.h)

Ім'я функції	*Загальний опис*
`time`	Повертає поточний календарний час системи типу `time_t`, як час від початку епохи Unix (від 01.01.1970) https://en.cppreference.com/w/c/chrono/time
`gmtime`, `gmtime_r`, `gmtime_s`	Перетворює час від початку епохи Unix на календарний час, представлений як універсальний координований час (Coordinated Universal Time, UTC) https://en.cppreference.com/w/c/chrono/gmtime
`localtime`, `localtime_r`, `localtime_s`	Перетворює час від початку епохи Unix на календарний час, виражений як місцевий час https://en.cppreference.com/w/c/chrono/localtime
`asctime_s`	Перетворює дані, що описані типом `tm` на текстове представлення https://en.cppreference.com/w/c/chrono/asctime
`ctime_s`	Перетворює дані, що описані типом `time_t` на текстове представлення https://en.cppreference.com/w/c/chrono/ctime

Повний перелік типів даних, функцій, макроконстант, які пов'язані з часом, можна подивитися за посиланням: https://en.cppreference.com/w/c/chrono.

Слід зауважити, що в *time.h* є функції, які в нових редакціях стандарту мови С були оголошені застарілими (deprecated) та з часом зовсім можуть бути вилучені з бібліотеки. Наприклад, це такі функції як: `asctime()` та `ctime()`. При використанні подібних функцій в різних прикладах слід уникати їх використання та замінювати їх виклики на більш удосконалені варіанти.

Таблиця Б.7

Бібліотечні функції для перевірки широких символів та їх перетворення (файл заголовку wctype.h)

Ім'я функції	*Загальний опис*
`iswalnum`	Перевірка на букву або цифру https://en.cppreference.com/w/c/string/wide/iswalnum
`iswalpha`	Перевірка на букву https://en.cppreference.com/w/c/string/wide/iswalpha
`iswcntrl`	Перевірка на символ керування https://en.cppreference.com/w/c/string/wide/iswcntrl
`iswdigit`	Перевірка на цифру десяткової системи числення https://en.cppreference.com/w/c/string/wide/iswdigit
`iswxdigit`	Перевірка на цифру шістнадцяткової системи числення https://en.cppreference.com/w/c/string/wide/iswxdigit
`iswgraph`	Перевіряє, чи має певний символ графічне представлення https://en.cppreference.com/w/c/string/wide/iswgraph

`iswlower`	Перевіряє, чи є певний символ у нижньому регістрі https://en.cppreference.com/w/c/string/wide/iswlower
`iswupper`	Перевіряє, чи є певний символ у верхньому регістрі https://en.cppreference.com/w/c/string/wide/iswupper
`iswprint`	Перевіряє, чи є певний символ символом для друку https://en.cppreference.com/w/c/string/wide/iswprint
`iswpunct`	Перевіряє, чи є певний символ символом пунктуації https://en.cppreference.com/w/c/string/wide/iswpunct
`iswspace`	Перевіряє, чи є певний символ пробільним символом https://en.cppreference.com/w/c/string/wide/iswspace
`iswblank`	Перевіряє, чи є певний символ пробілом або горизонтальною табуляцією https://en.cppreference.com/w/c/string/wide/iswblank
`towlower`	Повертає код символу для нижнього регістру https://en.cppreference.com/w/c/string/wide/towlower
`towupper`	Повертає код символу для верхнього регістру https://en.cppreference.com/w/c/string/wide/towupper

Таблиця Б.8

Бібліотечні функції для роботи з широкими рядками
(файл заголовку wchar.h)

Ім'я функції	*Загальний опис*
`wcscat`, `wcscat_s`	Конкатенація (об'єднання) двох рядків https://en.cppreference.com/w/c/string/wide/wcscat
`wcsncat`, `wcsncat_s`	Додає n-символів до рядку https://en.cppreference.com/w/c/string/wide/wcsncat
`wcschr`	Пошук спочатку рядка першого входження заданого символу https://en.cppreference.com/w/c/string/wide/wcschr
`wcsrchr`	Пошук з кінця рядка першого входження заданого символу https://en.cppreference.com/w/c/string/wide/wcsrchr
`wcsstr`	Пошук підрядку в рядку символів https://en.cppreference.com/w/c/string/wide/wcsstr
`wcscmp`	Порівняння двох рядків з врахуванням регістру символів https://en.cppreference.com/w/c/string/wide/wcscmp
`wcsncmp`	Порівняння двох рядків з врахуванням регістру символів та вказаної кількості символів https://en.cppreference.com/w/c/string/wide/wcsncmp
`wcscpy`, `wcscpy_s`	Копіювання одного рядку в інший https://en.cppreference.com/w/c/string/wide/wcscpy
`wcsncpy`, `wcsncpy_s`	Копіювання одного рядку в інший з врахуванням кількості символів https://en.cppreference.com/w/c/string/wide/wcsncpy
`wcslen`, `wcsnlen_s`	Повертає довжину рядка https://en.cppreference.com/w/c/string/wide/wcslen
`wcstok`, `wcstok_s`	Знаходить наступний токен у рядку з врахуванням певних параметрів https://en.cppreference.com/w/c/string/wide/wcstok
`fputwc, putwc`	Виведення символу у вказаний потік https://en.cppreference.com/w/c/io/fputwc
`putwchar`	Виведення символу у потік stdout https://en.cppreference.com/w/c/io/putwchar
`fgetwc, getwc`	Читання символу із вказаного потоку

	https://en.cppreference.com/w/c/io/fgetwc
`getwchar`	Читання символу із потоку stdin https://en.cppreference.com/w/c/io/getwchar
`wprintf`, `fwprintf`, `swprintf`, `wprintf_s`, `fwprintf_s`, `swprintf_s`, `snwprintf_s`	Форматоване виведення даних з використанням широких символів https://en.cppreference.com/w/c/io/fwprintf
`wscanf`, `fwscanf`, `swscanf`, `wscanf_s`, `fwscanf_s`, `swscanf_s`	Форматоване введення даних з використанням широких символів https://en.cppreference.com/w/c/io/fwscanf
`fputws`	Виведення в певний файловий потік широкого С-рядку https://en.cppreference.com/w/c/io/fputws
`fgetws`	Введення широкого С-рядку з певного файлового потоку з визначеною кількістю символів https://en.cppreference.com/w/c/io/fgetws

Таблиця Б.9

Бібліотечні функції підтримки локалі (файл заголовку locale.h)

Ім'я функції	*Загальний опис*
`setlocale`	Отримує та встановлює поточну локаль https://en.cppreference.com/w/c/locale/setlocale
`localeconv`	Повертає числові та грошові деталі форматування поточної мови https://en.cppreference.com/w/c/locale/localeconv

В *locale.h* також присутні певні макроконстанти (наприклад: LC_ALL, LC_NUMERIC та інші), а також описана структура даних `lconv`, в полях якої представлені деталі форматування числових та грошових одиниць поточної мови.

Таблиця Б.10

Підтримка перевірки на твердження (файл заголовку assert.h)

Ім'я макрофункції	*Загальний опис*
`assert`	Якщо макрос NDEBUG невизначений, то макрофункція перевіряє вказане твердження на рівність нулю і якщо це так, то виводить діагностичну інформацію на консоль та аварійно перериває виконання програми. Якщо макрос NDEBUG визначений в коді перед підключенням *assert.h*, то макрофункція не буде виконана https://en.cppreference.com/w/c/error/assert

Слід зауважити, що макрос NDEBUG не визначено стандартною бібліотекою мови С. Тобто пропонується приймати рішення чи визначати його, наприклад, в коді, чи ні, особисто розробнику. Реалізація `assert()` в Microsoft CRT дещо відрізняється від запропонованого стандартом мови С хоча й відповідає наведеному опису.

Навчальне видання

Гаркуша Ігор Миколайович

ПРОГРАМУВАННЯ

Методичні рекомендації до виконання лабораторних робіт
для здобувачів ступеня бакалавра освітньо-професійних програм
«Інформаційні системи та технології»
спеціальності 126 Інформаційні системи та технології
та «Комп'ютерна інженерія»
спеціальності 123 Комп'ютерна інженерія

Видано в авторській редакції.

Електронний ресурс.
Підписано до видання 28.04.2025. Авт. арк. 4,8.

Національний технічний університет «Дніпровська політехніка».
49005, м. Дніпро, просп. Дмитра Яворницького, 19.