Київський національний університет
імені Тараса Шевченка

Кононов М.В.

Програмування

Методичні вказівки
до лабораторних робіт

для студентів спеціальності
G5 «Електроніка, електронні комунікації,
приладобудування та радіотехніка»
факультету радіофізики, електроніки
та комп’ютерних систем

Київ – 2026

УДК 004.4:076

Програмування. Методичні вказівки до лабораторних робіт для студентів спеціальності G5 «Електроніка, електронні комунікації, приладобудування та радіотехніка» факультету радіофізики електроніки та комп'ютерних систем / Кононов М.В. – Київ: ФРЕКС Київського національного університету імені Тараса Шевченка, 2026. – 71 с.

Методичні вказівки містять завдання для виконання здобувачами практичної складової курсу «Програмування», який викладається студентам спеціальності G5 факультету радіофізики електроніки та комп'ютерних систем.

Метою дисципліни є ознайомлення студентів з базовими принципами програмування та синтаксисом однієї з мов. Основним завданням практичної складової є отримання навичок з розробки та налагоджування програм початкового рівня. Обрана в даному курсі мова програмування визначається специфікою спеціальності, як використовувана для мікроконтролерів та вбудованих систем.

Рецензенти:

доктор технічних наук, професор, заслужений діяч науки і техніки, старший науковий співробітник ДУ «Інститут медицини праці імені Ю.І. Кундієва Національної академії медичних наук України», професор кафедри штучного інтелекту НТУ «Київський політехнічний інститут імені Ігоря Сікорського» Литвиненко В.І.,

доктор технічних наук, доцент кафедри радіотехніки та радіоелектронних систем Київського національного університету імені Тараса Шевченка, Ольшевський С.В.

Ухвалено Вченою радою
факультету радіофізики електроніки та комп'ютерних систем
Київського національного університету
імені Тараса Шевченка
(протокол № 11 від 11.06.2026)

Зміст

Вступ

Відповідно до своєї назви спеціальність G5 стосується інженерії, пов’язаної з розробкою та експлуатацією електронних і радіотехнічних пристроїв, систем електронної комунікації (на жаль, цей термін замінив раніше використовуваний і актуальний в більшості європейських мов термін «телекомунікації») та вимірювальних приладів. Отже – пасивні та активні електронні компоненти, лінії передачі, антени, вимірювальні перетворювачі. Нібито програмування для усього цього є чимось зайвим і повинне залишитись іншим фахівцям, для яких є спеціальність F2 «інженерія програмного забезпечення». Але…

Час виключного використання аналогової електроніки давно у минулому. Навіть холодильники та пральні машини, не кажучи вже про керування системами автомобіля або авіоніку літаків, давно використовують програмовані пристрої. Щодо приладобудування, навіть звичний для інженерів з електроніки тестер за основну складову частину має мікроконтролер. Саме використання автоматики на основі програмного керування в багатьох випадках дозволяє суттєво збільшити точність вимірювань завдяки залученню більш досконалих, але складних для виконання людиною, алгоритмів.

Щодо радіотехніки, просто як приклад – для формування необхідних характеристик діаграми спрямованості використовуються багатоелементні антени з складними алгоритмами керування.

Комунікації. Для обміну інформацією використовуються системи (телефонія, комп’ютерні мережі тощо) на основі цифрового кодування даних. Отже, навіть аналогові складові таких пристроїв та систем або прямим чином пов’язані з використанням програмних технологій, або, як мінімум, повинні враховувати сумісність з ними.

А ще слід пам’ятати про активне використання математичного (комп’ютерного) моделювання, яке в останні десятиріччі суттєво збільшили ефективність роботи в інженерії. Дійсно – в такому моделюванні часто використовуються готові програмні продукти, при роботі з якими необхідно задати тільки конфігурацію моделі. Але, як мінімум, інженер повинен розуміти як такі моделі працюють і чому в деяких випадках з’являються не зовсім адекватні результати. Відповідно, для подолання певних проблем доводиться братися за написання коду та копирсання у математиці чисельних методів.

Дисципліна «Програмування» має відповідати своїй назві. Отже, її головною метою є ознайомлення з базовими принципами програмної обробки даних, засвоєння сучасних технологій проєктування програм. Певна увага приділяється програмуванню задач математичного моделювання та основними ідеями наближених чисельних методів.

Зауважимо, що програмне забезпечення поділяють на системне, яке забезпечує роботу інших програм, і прикладне, призначене для виконання конкретного завдання користувача. Розробка системних програм, на кшталт драйверів, хоча в певних випадках і це є актуальним для електронних комунікацій та приладобудування, є досить складним і вимагає високої кваліфікації. Відповідно, обмежимось тільки прикладними програмами.

Як мова програмування, далі використовується C++, чому – буде пояснено дещо пізніше. Але одразу зазначимо, що при вивченні програмування головним є не вибір конкретної мови, а розуміння самої ідеології обробки даних та керування на основі послідовності дій. В перспективі за необхідності перейти на іншу мову не так вже й складно.

Оскільки програмування є значною мірою практичною діяльністю, навіть лекційна складова цієї дисципліни вже насичена досить великою кількістю демонстраційних фрагментів програм, які ілюструють не тільки синтаксис мови програмування, а й особливості використання певних її елементів, структуризацію проєкту, деякі корисні прийоми роботи тощо. Практична ж складова включає в себе послідовність завдань, які є паралельними до лекційної і дозволяють поглибити відповідні (умовно теоретичні) знання у формі навичок. Отже, найбільш ефективним є вчасне виконання кожного із завдань одразу після відповідної лекційної теми, а зразки проєктів, які додаються до лекційного матеріалу, та слайди лекційних презентацій слушно використати як певні прототипи в процесі розв’язання практичних завдань.

Оскільки це програмування, *виконання завдань не вимагає оформлення окремого звіту на кшталт лабораторних з інших дисциплін. Результатом роботи повинна бути програма, яка (бажано) максимально повно виконує поставлене завдання та вміння відповісти на питання, а також показати, що виконує кожен з її рядків.*

1. Розробка проєкту прикладної програми

Для безпосереднього виконання обробки інформації обчислювальним пристроєм (процесором або мікро-контролером) програма повинна бути реалізована на основі машинних кодів – інструкцій, множина яких закладається при розробці архітектури такого пристрою. Ці коди є бінарними числами, оскільки з врахуванням технологічної доцільності обчислювальний пристій реалізує обробку саме такого типу даних, але для людини, яка розробляє програму, навпаки – числовий спосіб кодування є критично незручним.

Полегшити ситуацію може використання програмування низького рівня – мови асемблеру, яка використовує однозначну відповідність до кожної цифрової машинної команди зрозумілого для програміста слова. Основними недоліками застосування асемблера є сувора відповідність мови до системи команд конкретного процесора та складність для людини швидкого розуміння самого тексту програми, оскільки логіка побудови послідовності інструкцій мало нагадує те, як вона відтворюється в інших видах діяльності людей.

Якщо розглянути роботу мікроконтролера, то зазвичай він постійно виконує тільки одну програму, що дозволяє обмежитись інструкціями такої програми. На відміну від цього комп’ютер вимагає сумісної роботи багатьох програм з можливістю оперативного їхнього запуску та зупинки. Це вимагає спеціального прошарку між апаратною складовою комп’ютера та прикладною програмою – операційної системи. Для виконання програм операційна система має спеціальне середовище виконання – бібліотеки, ресурси пам'яті, логічні інтерфейси (специфікація взаємодії програмних компонентів), тощо. Разом зі специфікою системи команд процесора утворюється деяка програмно-апаратна платформа, особливості якої повинні бути враховані при розробці прикладної програми. Асемблер не вирішує цю проблему та застосовується, і то не дуже часто, тільки на рівні окремих включень у загальний проєкт комп’ютерної програми. І навіть при розробці програм для мікроконтролерів програмісти стараються відмовитись від глобального використання асемблера, залишаючи його фрагментам обслуговування тільки деякої критичної складової проєкту.

Для спрощення розробки програми та вирішення проблеми (як мінімум частково) незалежності від платформи були

розроблені певні мови високого рівня, орієнтовані більше на програміста, ніж на процесор. Отже, написаний високорівневою мовою текст для виконання повинен бути перекодований у послідовність машинних кодів. Таке перекодування виконує спеціальна програма – транслятор. Найпопулярнішими режимами трансляції є:

- в об'єктний код – бінарний код для усіх фрагментів проєкту, з набору яких компонувальник (редактор зв'язків) збирає готовий виконуваний модуль або бібліотеку в залежності від конфігурації проєкту;
- в байт-код – машинонезалежний код низького рівня, що виконується віртуальною машиною (специфічним інтерпретатором цієї «проміжної мови»);
- інтерпретація – пооператорна обробку первинного тексту програми безпосередньо при її виконанні (без попередньої підготовки).

Зрозуміло, що як будь-яка програмна обробка даних, трансляція вимагає обчислювальних ресурсів, особливо, коли потрібно виконувати синтаксичній розбір тексту, написаного в першу чергу для розуміння людиною. Через це інтерпретація є найповільнішим з огляду на виконання програми способом трансляції. Використання байт-коду є вже набагато швидшим, оскільки він значною мірою адаптований для фінальної «дотрансляції». Але повна відсутність потреби у програмі-інтерпретаторі та самого процесу такої додаткової обробки під час виконання забезпечує помітну перевагу в швидкодії та компактності «повністю компільованих» програм. Реалізація екосистеми C / C++ відповідає саме повній компіляції і завдяки цьому така, доволі стара, мова не втрачає популярності при програмуванні мікроконтролерів і у випадках, коли потрібне максимальне використання швидкісних можливостей процесора.

Створення програми передбачає реалізацію певної послідовності дій процесора. Одразу зазначимо, що не будь-яка послідовність дій відповідає усім ознакам алгоритму і не слід її надто вільно називати таким чином. Отже, для розробки програми необхідно виконати такі етапи:

- розробку самої послідовності дій програми (може завершуватись створенням графічної блок-схеми);
- написання тексту використанням однієї з мов програмування (результатом є один чи декілька текстових файлів);

- компіляцію текстових файлів з первинним кодом (при умові, що вона можлива, тобто немає критичних помилок) – завершується створенням об'єктних файлів;
- компоновку на основі програмної обробки об'єктних файлів – якщо успішно, результатом є виконуваний файл або бібліотека.

Перший з цих етапів технологічно відокремлений від наступних і є найбільш творчою роботою людини (якщо не застосовується штучний інтелект). Саме він прямим чином відноситься до проєктування. Сукупність інших етапів є більш «механістичною». Тому написання тексту, тобто переклад довільного опису проєкту на одну з мов часто називають кодуванням. Оскільки це робота з текстом, її можна виконати за допомогою будь-якого текстового редактора, навіть включеного в довільний офісний пакет. Але при цьому є певні нюанси.

Останні два пункти виконують спеціальні програми (транслятор та компоновник), тобто не вимагають від програміста окремих зусиль, крім їхнього запуску, що можна зробити командним рядком. До речі, такий символьний інтерфейс запуску програм використовувався багато років і не втратив актуальності. Однак в більшості випадків більш зручним є використання комбінованого продукту, що автоматизує процес та зменшує обсяг рутинної роботи програміста об'єднанням багатьох функцій у єдину систему. Клас таких продуктів має назву «Інтегроване середовище розробки» або IDE (англ. integrated development environment). Зазначимо, що є IDE з вбудованим компілятором під певну програмно-апаратну платформу або з перемиканням між декількома, а є пакети, які передбачають додаткове встановлення певних засобів трансляції за бажанням програміста (можуть постачатись взагалі без компілятора).

IDE часто реалізують в архітектурі оболонки, яка забезпечую зручну взаємодію з людиною через оптимізований певним чином графічний інтерфейс користувача (англ. GUI, graphical user interface) та виклик усіх функцій через його елементи. При цьому вбудований в пакет редактор тексту (саме завдяки тому, що він вбудований) має додаткові можливості:

- виведення пов'язаних з відповідними рядками первинного тексту позначок синтаксичних помилок, які знаходить транслятор, та повідомлення про наявність проблем компоновки;

- змістове виділення кольорами частин тексту програми (це зручно !);
- підказки можливого продовження при наборі на клавіатурі зарезервованих слів мови програмування, ідентифікаторів тощо.

Отже, розробник проєктує задачу, кодує її та виправляє можливі синтаксичні помилки. В результаті програма успішно компілюється та може бути запущена на виконання. І що – Її вже можна передавати замовнику (або показувати викладачу для отримання оцінки)? Звичайно ж – ні. Якщо програма складніша за декілька рядків, скоріше за все перше виконання буде зовсім не таким, як планувалось. Щоб заставити програму виконуватись правильно, необхідно виконати ще один доволі складний та довгий процес проєктування – налагодження або зневадження (англ. Debugging).

В історії розвитку програмних технологій був період, коли для налагодження програм не було спеціальних засобів. Для пошуку помилок часу виконання використовували додаткове виведення самою програмою спеціально включеними в неї командами, що виводили текстові повідомлення та значення окремих змінних. З часом з'явились спеціальні засоби налагодження, які забезпечили значно кращу ефективність та зручність його виконання за рахунок:

- можливості проходження програми крок за кроком;
- виконання програми до спеціально встановленої точки зупину (англ. Breakpoint) – щоб не надто довго усе проходити покроково;
- наявності спеціального інтерактивного інструментарію відстеження значень змінних.

Є й інші корисні для розробника інструменти, на яких ми не зупиняємось, оскільки вони використовуються у професійній роботі з проєктами підвищеної складності, тобто виходить за межі даного курсу. Налагоджувач тісно взаємодіє з вбудованим в IDE текстовим редактором, демонструючи в тексті поточне місце виконання, що також збільшує зручність роботи.

Налагодження програми часто вимагає більших витрат часу та уважності, ніж написання тексту та первинне виправлення синтаксичних помилок. Отже, *в практичній складовій курсу з програмування, в тому числі при оцінюванні результатів виконання завдань, приділяється надзвичайно*

велика увага вмінню використанням налагоджувача довести програму до бажаного результату.

Часто традиційна послідовність вивчення програмування починається із засобів введення з клавіатури та виведення у вікно консольного застосунку. Але в реальних проєктах кожен з виконавців розробляє свою частину проєкту, а взаємодія з іншими його складовими забезпечується зовсім не через користувача-посередника, який буде виконувати ввід та вивід даних. Для цього використовується безпосередній виклик функції (якщо тексти окремих частин пізніше об'єднуються у спільний проєкт), взаємодія через файли або сокети тощо.

Спосіб обміну даними узгоджується ще на етапі розподілу завдань між окремими програмістами і вони повинні дотримуватись домовленості. В цьому випадку використання консолі є не просто зайвим, а шкідливим. Більше того, навіть введення вхідних даних та виведення кінцевих результатів у консольне вікно зазвичай є залишком «технологій кам'яного віку» – для цього краще використовувати файли. І пам'ятаємо про те, що сучасні програми для взаємодії з користувачем взагалі використовують графічний інтерфейс і, відповідно, зовсім інші функції вводу-виводу. А ще є мікроконтролери – де там консоль?! Саме тому *при виконанні практичних завдань у цьому курсі використання вводу з клавіатури та виводу в консольне вікно, крім спеціально зазначених випадків, є забороненим.* Так, іноді консоль є корисною, але не так вже й часто. В інших випадках при розробці програми треба використовувати засоби налагоджування, а задати певним змінним необхідні значення можна оператором присвоєння у тимчасових рядках, які при втраті актуальності зазвичай навіть не видаляють, а з оглядом на потребу налагоджування в майбутньому «закоментовують», тобто вилучають з обробки сполученням символів для коментування («//» або «/*» … «*/»).

При виконанні практикуму рекомендується використовувати інтегрований засіб розробки, що застосовується на лекціях (Code::Blocks), як альтернативу https://www.onlinegdb.com/online_c++_compiler – досить зручний онлайн IDE. Його перевагою є відсутність потреби інсталювати, але має місце і недолік – дещо менша зручність. Хоча за бажанням можна використовувати й інші пакети розробки.

2. Описи лабораторних робіт

Лабораторна робота №1. Використання IDE

Мета роботи: Навчитись вводити та редагувати текст програми, шукати та виправляти прості помилки, компілювати та запускати програму, виконувати прості дії з перевірки її працездатності та виконувати нескладні зміни функціональності.

Загальні відомості

При виконанні цієї роботи поки що не вистачає базових знань із синтаксису мови програмування, але дещо вже можна зробити. Робота у текстовому редакторі IDE є досить близькою за основними діями користувача до аналогічного компонента офісного пакету – традиційні методи та інструменти перегляду тексту, його редагування та пошуку майже співпадають. Тобто попередня підготовка тексту програми може вже бути виконана і з підготовкою на рівні вступної лекції.

Найцікавішим є спробувати певними діями дослідити (навіть не знаючи мови програмування), який рядок програми що виконує і за рахунок цього спробувати визначитись з деякими синтаксичними структурами. Особливо, якщо взяти до уваги, що мова програмування будується так, щоб текст програми трохи нагадував речення англійською. Отже, при виконанні цього завдання реалізується своєрідна «зворотна розробка» (англ. reverse engineering), тобто виконання дослідження готової програми для розуміння її принципів роботи з метою створення схожої.

При дослідженні поведінки програми слід змінювати окремі параметри у певному рядку та після перекомпіляції та запуску програми дивитись на відмінності у її виконанні. Для цього зручно використати вбудовані в IDE інструменти налагодження програми.

Особливу увагу в даному завданні слід звернути на повторювані дії, для виконанні яких використовуються так звані цикли. Крім того, оскільки в даному випадку виконується символьне виведення на консоль, слід також дивитись на особливості кожного виведення. Ось той випадок, коли у завданні використовуємо консольне виведення, але в даному випадку таке виведення і є метою досліджуваної в цій лабораторній роботі програми.

Завдання

1. Ознайомитись з основними функціями IDE Code::Blocks.
2. Навчитись основним діям при розробці програми.
3. Використовуючи деякий зразок програми реалізувати власну із схожими функціями.

Хід виконання

1. Основні функції Code::Blocks

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_01 у вказаній викладачем папці.

1.2. Ознайомитись з автоматично створеним початковим текстом програми, скомпілювати, запустити на виконання (у режимі налагодження), подивитись результат роботи у вікні консолі.

1.3. Поставити точку переривання, запустити (у режимі налагодження), пересвідчитись у виконанні зупинки. Вийти з налагодження.

1.4. Визначити порядковим проходженням, який рядок відповідає за виведення тексту у вікно консолі.

1.5. Переробити текст програми за зразком на рис.1.1.

1.6. Досягти коректної компіляції програми пошуком та виправленням можливих при введенні тексту помилок.

1.7. Запустити на виконання (у режимі налагодження), подивитись результат роботи у вікні консолі.

1.8. Використовуючи точки переривання та покрокове проходження подивитись, який з рядків що виконує.

Для перегляду стану змінних активуйте відповідне вікно (коли виконання програми зупинилось на точці переривання):

Debug -> Debugging windows -> Watches

1.9. Показати працездатність програми викладачу.

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

int main()
{
    int Min=2;
    int Max=10;

    printf("\n");

    for (int y=Min; y<Max; y++)
    {
        for (int x=Min; x<Max; x++)
        {
            int mult=x*y;
            printf("%d x %d = %d \n", x, y, mult);
        }
        printf("-------------\n");
    }
    printf("\n\n\n");
//-----------------------------------
    system("pause");
}
```

Рис.1.1

2. Створення програми

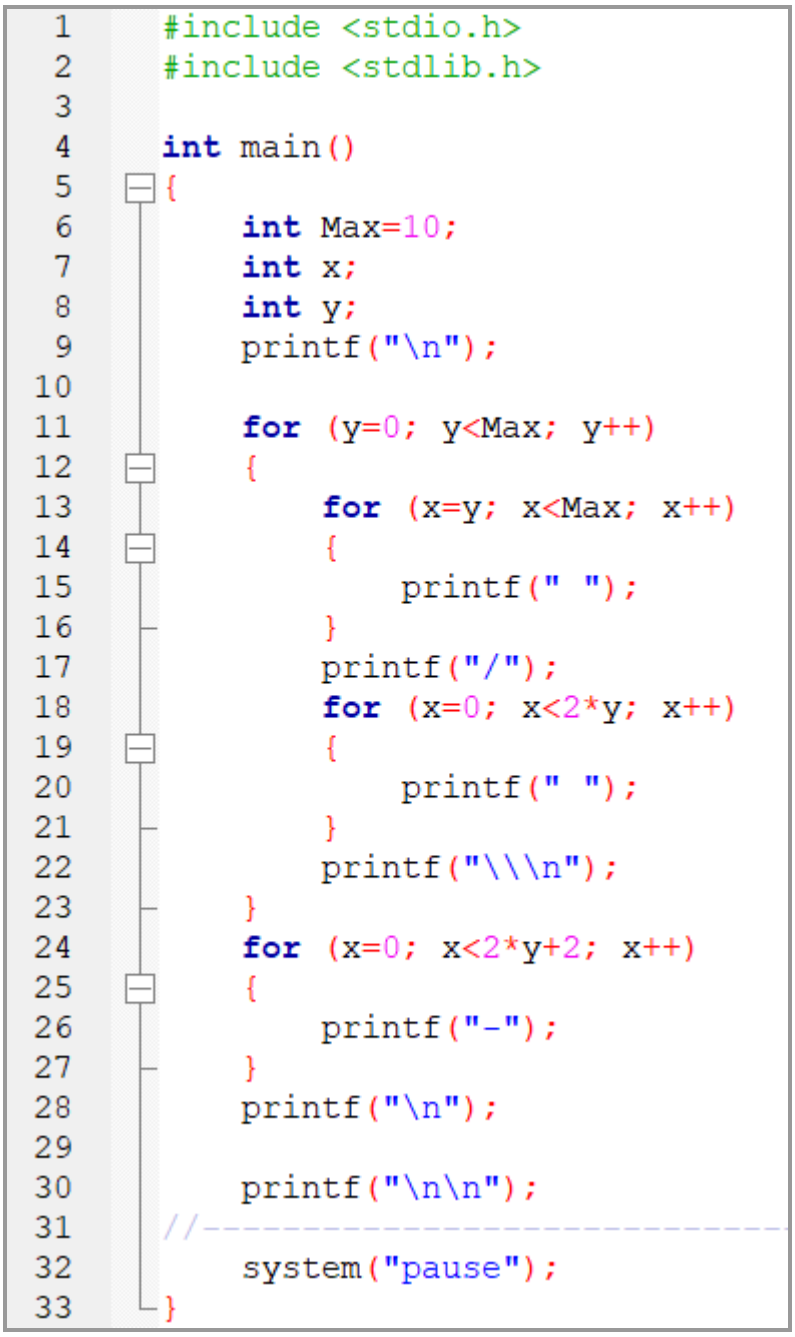
2.1. Закрити Code::Blocks. Заново запустити Code::Blocks та відкрити раніше створений (вже існуючий) проєкт.

2.2. Переробити текст програми за зразком на рис.1.2. Навчитись економним методам роботи з текстом на снові копіювання та редагування тексту.

2.3. Добитись коректної компіляції програми пошуком та виправленням можливих при введенні тексту помилок. Програма повинна формувати у вікні консолі трикутник (рис.1.3).

2.4. Використанням точок переривання та покрокового проходження подивитись, який з рядків що виконує.

2.5. Показати працездатність програми викладачу.



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

int main()
{
    int Max=10;
    int x;
    int y;
    printf("\n");

    for (y=0; y<Max; y++)
    {
        for (x=y; x<Max; x++)
        {
            printf(" ");
        }
        printf("/");
        for (x=0; x<2*y; x++)
        {
            printf(" ");
        }
        printf("\\\n");
    }
    for (x=0; x<2*y+2; x++)
    {
        printf("-");
    }
    printf("\n");

    printf("\n\n");
//------------------------------
    system("pause");
}
```

Рис.1.2

3. Створення власної програми

3.1. Змінити текст програми так, щоб виведення на консоль отримало вигляд, як на рис.1.4.

3.2. **Додаткове**: Змінити текст програми так, щоб виведення на консоль отримало вигляд, як на рис.1.5. (простіше завдання) або рис.1.6 (складніше).

3.3. Показати працездатність програми викладачу.

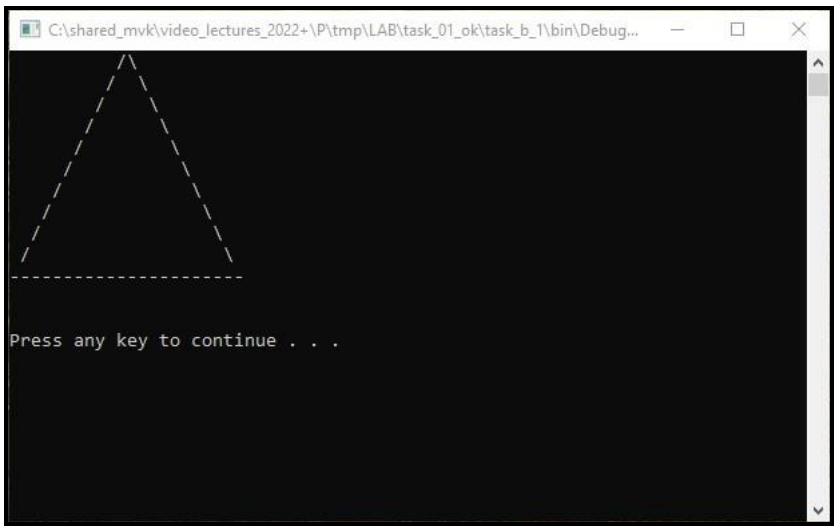


Рис.1.3

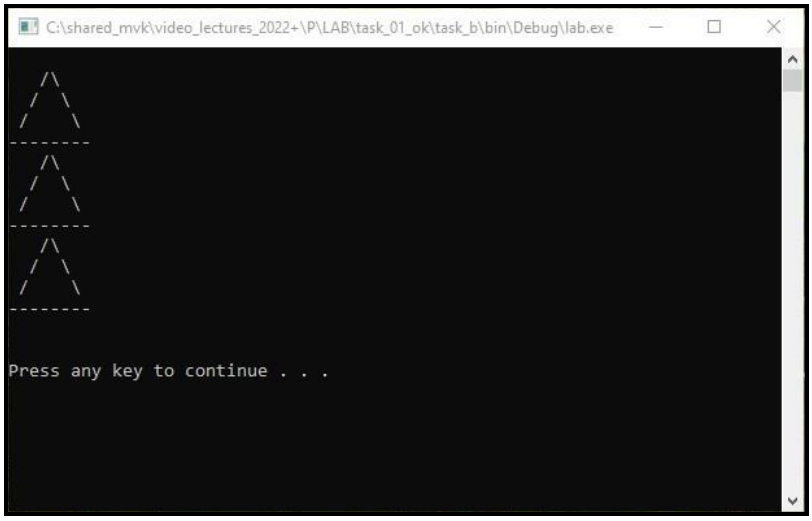


Рис.1.4

Рис.1.5

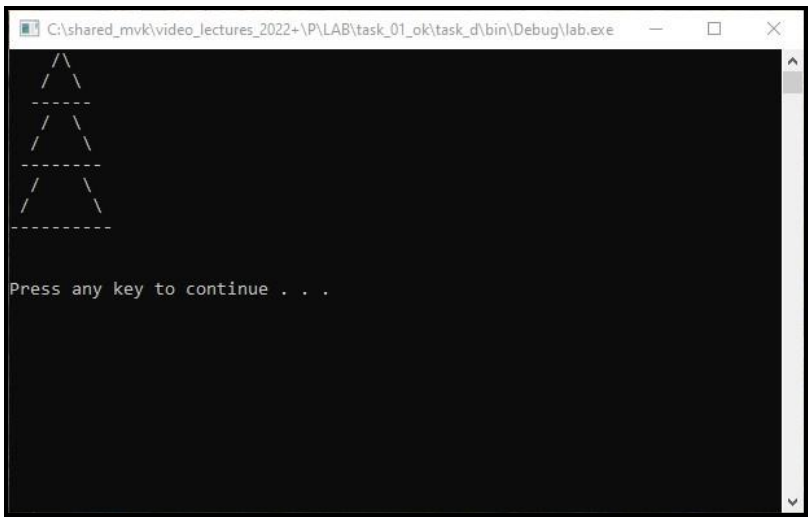


Рис.1.6

Виконання даного завдання дозволило познайомитись з послідовністю основних дій, що виконуються при розробці програми у єдиному комбінованому засобі розробки та відповідними інструментами такого засобу. Крім того, це завдання дозволило спробувати на практиці *базові принципи налагоджування програми:*

- *покрокове проходження* дозволяє визначити правильність виконання порядку дій програмою – особливо корисним це буде, коли будете детальніше знайомитись з розгалуженнями, циклами, викликом функцій;
- *точки переривання* дозволяють виконати швидке виконання фрагментів коду, які в даний момент не має сенсу «проклацувати» покроково;
- основним способом побачити, що саме виконує програма в певному її місці, є *перегляд значень змінних та як вони змінюються*.

Сподіваємось, що правильне використання налагоджувача спростить виконання інших завдань.

Лабораторна робота №2. Арифметичний вираз

Мета роботи: Навчитись використовувати опис змінних та арифметичні вирази на прикладі розв'язування простої математичні задачі.

Загальні відомості

Довільна комп’ютерна програма обробляє завдання на основі певної послідовності арифметичних та логічних операцій. До речі, саме тому основним пристроєм обробки даних є арифметико-логічний пристрій. І навіть при обробці інших видів даних вони зводяться кодуванням до числового представлення і обробляються тими ж діями. Коли для цього використовується виклик бібліотечної функції, яка нібито не виконує прямі розрахунки, слід розуміти, що вони просто приховані всередині функції. Якщо відволіктись від програмування, людина при виконанні подібних задач діє так само. Порівняйте, наприклад, із задачами з курсу середньої школи.

У програмуванні однією з базових синтаксичних конструкцій є вираз (expression) – комбінація змінних, констант, операторів та викликів функцій, з якими виконуються арифметичні або логічні дії. Відповідно до того, результатом є число або булеве значення, вирази й діляться на арифметичні та логічні. Подібні вирази ми використовуємо і без комп’ютера при розв’язанні багатьох задач. Отже, при перенесенні задачі на комп’ютерне виконання принципи використання виразів і значна частина їхнього синтаксису зберігається – є операнди, що на момент виконання дій мають певні значення, та дії, що виконуються з ними, а для зручності написання комбінації операцій, що виконуються послідовно, вони об’єднуються у спільний ланцюжок.

Як і при використанні виразів без комп’ютера, в результаті їхнього виконання на відміну від інструкцій (statement) обов’язковим чином отримується результат – як вже зазначалось, арифметичний, тобто число, або логічний (true / false). Для забезпечення можливості подальшого використання цього результату частіше за все його записують в деяку змінну оператором присвоєння.

Змінною в програмуванні називають спеціально виділену для збереження блоку даних ділянку пам'яті комп'ютера (певну кількість байтів). Для виділення конкретної змінної серед інших вона має ім'я (ідентифікатор), яке приписується при її описі. В

мовах зі статичною типізацією, до яких відноситься C / C++, при описі задається також тип даних – фактично, хоч і неявно, кількість байтів та спосіб інтерпретації бінарного коду, що в них зберігається. Той самий бінарний код може задавати, наприклад, ціле, беззнакове ціле або дійсне число. В процесі подальшого використання змінної її тип (знову таки – при використанні мови зі статичною типізацією) змінити не можна. Внесення даних у змінну дозволяє певний час їх зберігати для подальшого використання. І тут також є аналогія з діями людини – при виконанні обчислень без комп'ютера ми також кудись записуємо проміжні результати для використання через деякий час.

Зазначимо, що для виконання завдання 3 виникне потреба у повторюваних діях – буде необхідно перебрати по черзі усі елементи масиву. За такі дії відповідає оператор циклу. Один з варіантів його реалізації – for, вже знайомий за попередньою лабораторною роботою:

```
for (int n =1; n <  N; n++)
{
    // те, що повторюється в циклі
}
```

В цьому випадку змінна n приймає значення від 1 до N (саме «до», а не «включно з»), кожного разу збільшуючись на 1 (за це відповідає оператор інкременту ++). І ще – для порівняння двох значень використовується логічний оператор:

```
if(a < b)
{
   // якщо умова виконується
}
```

Завдання (варіант 1)

1. В основі піраміди лежить прямокутник зі сторонами a, b. Відомі кути до основи alpha, beta (задані у градусах!) двох протилежних прилеглих до ребра a бічних граней. Знайти об'єм піраміди (рис.2.1):

   a = 25;
   b = 20;
   alpha = 60;
   beta = 45.

2. Знайти площу поверхні піраміди, якщо відомий ще один кут бічної грані до основи, він дорівнює 45 градусам.
3. Розрахувати суму елементів data[n] масиву, знайти мінімальний елемент цього масиву та його номер:

double data[ ] ={21.16, 22.2, 21.11, 14.2, 35.29, 72.4, 11.12, 34.92, 27.34, 11.25 };

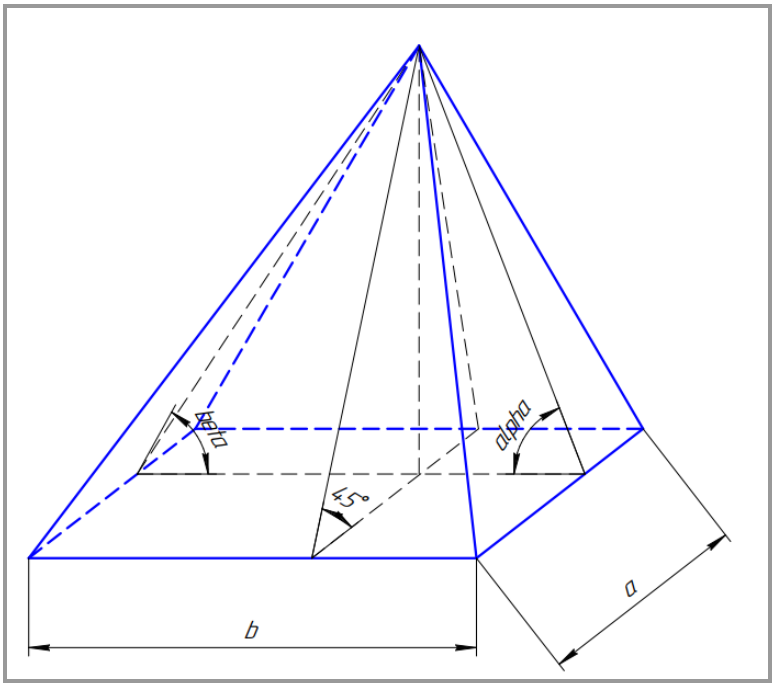


Рис.2.1

Завдання (варіант 2)

1. В основі піраміди лежить прямокутник зі сторонами a, b. Протилежні бічні грані є рівними. Висота піраміди h (рис.2.2). Знайти об’єм піраміди:

   a = 25;
   b = 20;
   h = 15.
2. Знайти площу поверхні піраміди.
3. Розрахувати добуток елементів data[n] масиву. Знайти кількість від'ємних елементів масиву та його максимальний елемент:

double data[ ] ={ 6.3, -1.42, -2.1, 4.5, 7.1, 0.2, -4.9, -0.03, 1.32 };

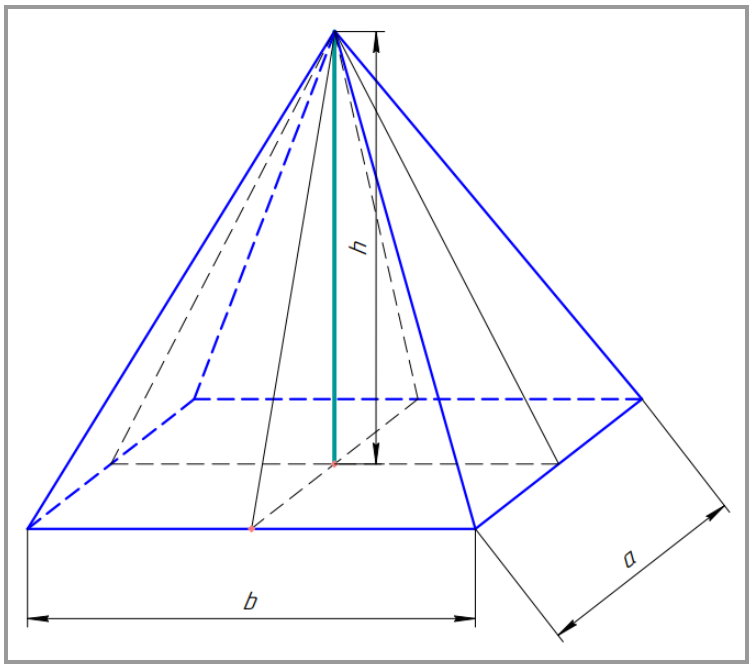


Рис.2.2

Хід виконання

1. *Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_02 у вказаній викладачем папці.*
2. *Розв'язати стереометричну задачу до рівня формул, які буде потрібно реалізувати в програмі.*

Для можливості виклику математичних функцій sin( ), cos( ), tan( ), sqrt( ) на початку програми додайте рядок:
#include <math.h>

3. *Реалізувати програмно п. 1 та 2. Показати результат викладачу.*
4. *Включити в програму цикл, що перебирає усі елементи заданого масиву.*
5. *Включити в цикл задану обробку елементів масиву. Показати результат викладачу.*

Отже, якщо вдалось виконати завдання у повному обсязі, ви побачили певну відповідність дій людини та комп'ютера при виконанні обчислювальної задачі на прикладі виразів та виконання нескладних повторюваних дій – в даному випадку вибору більшого або меншого значення з двох.

Лабораторна робота №3. Цикли та розгалуження

Мета роботи: Навчитись використовувати цикли та розгалуження на прикладі розв'язування простої математичні задачі.

Загальні відомості

Програми, довші за декілька рядків, зазвичай вимагають більш складного порядку виконання інструкцій, ніж проста їх лінійна послідовність, коли кожна інструкція виконується один раз, рядок за рядком, як написано у тексті програми. Зауважимо, що у будь-якій програмі є значна частина коду, що виконується саме так. Але у певних її місцях виникає ситуація, коли якусь дію чи комбінацію декількох дій необхідно виконувати тільки при певній умові, а інакше пропустити. Отже, лінійна послідовність не може бути використана, замість неї повинно бути виконання за одним з двох шляхів. Для цього використовується оператор розгалуження (умовний оператор) if. В такому, найпростішому, варіанті альтернативний шлях (коли умова не виконується) є порожнім. Якщо ж і його необхідно прописати, використовується конструкція if – else. В певних випадках (далеко не в усіх) для розгалуження на декілька гілок замість ускладненої конструкції з декількох if – else використовується оператор перемикача – switch, перевага якого є виключно у кращому сприйнятті тексту програми людиною.

Крім розгалуження в певних частинах завдання доводиться виконувати схожу обробку даних з деяким їх варіюванням, як вже мали можливість пересвідчитись при виконанні попередніх лабораторних робіт. Реалізувати повторювану сукупність дій копіпастом певного фрагменту коду декілька разів (як це можна було зробити з «ялинкою» з першої роботи) можна, хоч це виглядає не найкращим чином, і тільки коли кількість повторень є відомою та незмінною. Але у складніших випадках такий копіпаст є просто неприйнятним.

Оператор розгалуження разом з безумовним переходом goto, якщо його підтримує використовувана мова програмування (C++ підтримує), дозволяє організувати такі повторення, але вони виглядають досить кострубато на фоні досить непогано оптимізованої для сприйняття людиною мови високого рівня. Зручнішими є спеціальні оператори циклу. В C / C++ є декілька таких операторів, які майже в усіх випадках можуть бути замінені один одним, але є певні рекомендації, що випливають з

вимог зручності, в першу чергу для сприйняття людиною написаного коду:

- for використовують, коли точно відома кількість повторень (наприклад, є необхідність пройти по елементах масиву, що вже зустрічалось);
- while (та do – while) є зручним для завдання повторення доки виконується певна умова, яка може змінюватись у тілі циклу і визначається не певною логікою при написанні програми, а конкретними даними при виконанні готової програми.

Більш сучасні мови в тому чи іншому варіанті синтаксису підтримують ще й спрощений запис проходження по всіх елементах – foreach. Останні версії C++ також його мають – for (int x: arr). В рамках даного курсу цей оператор не розглядаємо з врахуванням того, що він принципово нічого не додає у логіку виконання, а тільки спрощує сам вигляд циклу.

Отже, при виконанні цієї лабораторної роботи знайомимось з тим, коли, а головне як, структурувати програму розгалуженнями та циклами у випадку трохи складнішого завдання, ніж ті, що були в попередніх лабораторних.

Щодо структуризації слід зупинитись ще на одному важливому моменті. *У програмуванні, як і взагалі в інженерії, прийнято виконувати проєкт саме так, як він прописаний у технічному завданні. Тобто завдання, в якому не передбачено розділення на окремі частини, слід виконувати у вигляді єдиного модуля (одного програмного проєкту).* Якщо ж ви раніше розбивали завдання на декілька проєктів у минулих лабораторних, прийшов час позбутись цієї хиби.

Відповідно, структуруємо програму в рамка одного проєкту. Якщо ви вже вмієте використовувати функції, можна кожне з підзавдань реалізувати окремою функцією і викликати ці функції з main. Альтернативою є послідовне включення відповідних фрагментів коду у main з позначенням їх текстовим поясненнями у коментарях (зазвичай це роблять перед відповідним фрагментом).

Як і в попередній роботі, при виконанні цього завдання є необхідність у повторюваних діях – на основі перебору усіх елементів масиву, але звертаємо увагу на те, що в даному випадку вимагається «подвійний» перебір для дослідження поведінки пари елементів замість одного. При цьому слід пам’ятати, що перестановка місцями елементів пари не змінює

саму пару – це треба врахувати при написанні відповідних операторів циклу.

Також слід звернути увагу на те, що досліджується умова тільки наявності відповідності пар першої множини до пар другої, тобто якщо для певної пари з першої множини одна така відповідність знайдена, для неї перевірку слід зупинити – згадайте, як зробити передчасний вихід з циклу.

Завдання (варіант 1)

1. Знайти скільки пар із заданої множини цілих чисел утворюють суму, яка дорівнює добутку довільних двох чисел з іншої заданої множини:

   int a[ ] ={15, 35, 12, 4, 9, 17, 47, 85, 15, 5, 19, 31, 7 ,11, 3};
   int b[ ] ={5, 11, 8, 6, 12 };

2. Розрахувати ряд (двома варіантами циклу):

$$\sum_{n=1}^{\infty} \frac{n^2}{n!}$$

   Умова завершення підсумовування – поточний доданок є меншим за 0.0001.

3. **Додаткове**: Знайти суму перших п'ятнадцяти двозначних простих чисел.

Завдання (варіант 2)

1. Знайти скільки пар із заданої множини дійсних чисел утворюють округлення суми, яке націло ділиться на один з елементів заданої множини цілих чисел:

   double a[ ] ={11.25, 32.2, 5.21, 54.1, 33.9, 14.7, 41.72, 39.95, 17.44, 61.05 };
   int b[ ] ={5, 11, 8, 6, 2, 12 };

2. Розрахувати ряд (двома варіантами циклу):

$$\sum_{n=1}^{\infty} \frac{n^2}{\sum_{m=1}^{n} m^3}$$

   Умова завершення підсумовування – поточний доданок є меншим за 0.0001.

3. **Додаткове**: Знайти суму перших двадцяти тризначних простих чисел.

Хід виконання

1. *Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_03 у вказаній викладачем папці.*
2. *Розв'язати завдання 1. Показати працездатність програми викладачу.*
3. *Розв'язати завдання 2. Показати працездатність програми викладачу.*

Це завдання було пов’язане з певним ускладненням структури програми. При правильному виконанні (без використання консольного вводу / виводу) була можливість додатково потренуватись у застосуванні налагоджувача при більш складній послідовності дій.

Реалізація послідовності інструкцій в межах одного блоку, що в мінімальному варіанті дозволяло впоратись з цією лабораторною, є малоприйнятним для ще складніших задач, які вимагають інших способів структуризації, розгляду яких присвячені наступні завдання.

Лабораторна робота №4. Функції

Мета роботи: Навчитись розробляти проєкт з декількох функцій (правильно описувати та викликати їх).

Загальні відомості

В багатьох задачах виникає ситуація, коли є необхідність виконати певну послідовність дій з різними даними, що формуються в різних місцях програмного коду. Знову копіпаст? Це збільшує загальний обсяг тексту програми і через це ускладнює її розуміння. Крім того, при потребі виправлення доводиться вносити зміни в декількох місцях. А ще збільшується обсяг оперативної пам'яті, яку займає програма при виконанні, хоч в сучасних умовах це в більшості випадків не дуже критично (виняток – програмування мікроконтролерів).

Для вирішення зазначеної проблеми ще на рівні асемблеру та початкових мов високого рівня реалізували механізм підпрограм та їх виклику. В певних мовах підпрограми розділяли на функції (вони повертають значення так, щоб його можна було використати, наприклад, оператором присвоєння) та процедури (не повертають значення). У C / C++, як практично у всіх сучасних мовах, використовують єдиний термін «функція» за рахунок того, що значення, що повертається, можна ігнорувати, або взагалі замість типу при описі функції задати void (позбавлений, вакантний).

Для написання тексту реалізації функції її тіло, тобто те що вона виконує, синтаксично виділяється (в C / C++ фігурними дужками), а перед цим задається рядок її опису. В цьому рядку як для змінних, щоб вирізняти з множини багатьох, задається ім'я (ідентифікатор), за яким у круглих дужках через кому йде список формальних параметрів – опис «вхідних» змінних з вказуванням імен та типів. Навіть коли цей список порожній, дужки присутні, як ознака для транслятора відмінності описів функції та змінної. Перед іменем вказується (якщо не void) тип, який повинне мати значення, що повертається через оператор return.

Дані у функцію в C / C++ та інших мовах, які дозволяють глобальні змінні, можуть передаватись та повертатись через них, але такий спосіб краще не використовувати, оскільки у складних програмах це збільшує плутанину та ймовірність помилок. Отже, використовуємо параметри. Виклик функції виконується через її ім'я, що супроводжується (також в дужках

через кому) списком фактичних параметрів – значеннями (їх можна задати літералами, змінними, константами, виразами), що присвоюються відповідним формальним параметрам. Оскільки це присвоєння, воно працює в один бік – у функцію. Але якщо передати у функцію не значення змінної, а її адресу через механізм вказівників (в C *name, в C++ опис формального параметра як посилання &name), то через такий параметр можна і щось повернути. Це дуже потужний механізм взаємодії з функцією.

```
bool F(int a, int b, double c, double &result)
{
        // . . .
        result = 0.;
        return true;
}

double res;
bool ok=F(10, 20, 10./3, res);
```

Будьте уважні – масиви у функцію передаються за посиланням (адресою його початкового елемента), тобто зміна в тілі функції значень елементів масиви буде дійсною за її межами. І ще – довжину масиву у функцію необхідно передати, бо в C / C++ немає способу всередині функції визначити це значення.

Завдання (варіант 1)

1. Реалізувати дві функції, що повертають максимальне та мінімальне з трьох дійсних значень, які передаються як параметри.
2. Реалізувати функцію, яка підсумовує максимальні з трійок значень (елементів трьох масивів з однаковим індексом). Для визначення максимального з трійки значень використати виклик функції з п.1.
3. Реалізувати ускладнену функцію, в якій розраховуються **дві** суми – максимальних та мінімальних з трійок значень (елементів трьох масивів з однаковим індексом). Для визначення максимального та мінімального з трійки значень використати виклик функції з п.1. Результати повернути через параметри.

Завдання (варіант 2)

1. Реалізувати дві функції – перша перевіряє ділимість націло першого параметра на хоча б один з інших двох параметрів, друга – визначає залишки від ділення першого параметра на інші два наступних, а результати повертає через параметри.
2. Реалізувати функцію, яка підсумовує кількість випадків кратності (п.1) для трійок значень (елементів трьох масивів з однаковим індексом).
3. Реалізувати ускладнену функцію, яка підсумовує **дві** суми – кількість випадків кратності (п.1) для трійок значень (елементів трьох масивів з однаковим індексом) та сум залишків від ділення (п.1). Результати повернути через глобальні змінні.

Хід виконання

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_04 у вказаній викладачем папці.

1.2. У функції main описати три масиви однакової довжини з довільними даними, їхній тип задати відповідно до варіанту завдання.

2. Функції, що виконують обробку даних, які передаються через параметри

Реалізацію усіх додаткових функцій робити **після** функції main.

2.1. Реалізувати першу з функцій завдання п.1.

2.2. Перевірити працездатність розробленої функції додаванням відповідних викликів з функції main. Що треба додати у проєкт крім реалізації функції для забезпечення можливості її виклику?

2.3. Реалізувати другу з функцій завдання п.1, використовуючи як зразок вже реалізовану, й також

перевірити її працездатність викликами з функції main.

2.4. **Додаткове:** При реалізації кожної функції у проєкті створити окрему пару файлів - source та header.

3. Функція, яка обробляє елементи масивів викликом функції з п.1 завдання

3.1. Реалізувати функцію, яка відповідно до завдання п.2. обробляє елементи масивів.

3.2. Перевірити працездатність функції викликом з функції main.

3.3. Показати працездатність функції викладачу.

4. Ускладнена функція, яка обробляє елементи масивів

4.1. Реалізувати ускладнену функцію, завдання п.3. Зверніть увагу на спосіб повернення результату, що вказаний у відповідному варіанті завдання.

4.2. Перевірити працездатність функції.

4.3. Показати працездатність функції викладачу.

Отже, якщо завдання виконане, познайомились з потужним інструментом структуризації програм – функціями. Тепер цей інструмент можна повноцінно використовувати, що і буде потрібне у наступних завданнях.

Зазначимо, що функції можуть використовуватись не тільки для багатьох викликів із різних місць програми, або декілька разів з різними значеннями параметрів, а з одноразовим викликом, тобто виключно для відокремлення певної частини коду з метою покращення розуміння архітектури складного проєкту при його перегляді.

Лабораторна робота №5. Функції +

Мета роботи: Вдосконалити навички розробки проєкту з декількома функціями.

Загальні відомості

В рамках подальшого тренування з використання функцій у програмуванні згадаємо, що термін «функція» зустрічається також у математиці. І багато з таких математичних функцій реалізовані у бібліотеках, які або входять в інсталяційний пакет IDE (як стандартна бібліотека певної мови програмування), або додатково приєднуються до розроблюваного проєкту. Багато широковживаних математичних функцій не можуть бути точно представлені певним арифметичним виразом на основі знайомих зі школи чотирьох дій, як, наприклад це можна зробити для зведення у цілий степінь. Отже, в комп'ютері – пристрої, який має певні проблеми з алгеброю (хоч і є програмні пакети, які непогано виконують алгебраїчні дії), доводиться змиритись з реалізацією цих функцій з певною похибкою, тобто наближено. Як мінімум, слід враховувати, що як вихідне значення, так і результати проміжних дій, є дійсними, а для його збереження комп'ютер використовує обмежену кількість розрядів дробової частини числа, точніше його мантиси. Вже це є причиною певної (хоч і невеликої) похибки.

Є різні способи реалізації математичних функцій наближеними числами. Одним з них є використання апроксимації рядами, наприклад рядом Тейлора – нескінченною сумою доданків, які обчислюються за значеннями похідних функції в певній точці. Нескінчений ряд є точною реалізацією, але в розрахунках є зазначене вище обмеження точності дійсних чисел, а головне – підсумовувати до нескінченості просто неможливо. Отже, доводиться обмежувати кількість доданків. Зрозуміло, що зменшення кількості доданків скорочує час обрахунків, але збільшує похибку обчислення через більшу відкинуту частину нескінченого ряду.

Загалом ряд в математиці за визначенням є результатом послідовного додавання нескінченної кількості величин та ззвичай задається деяке формальне правило визначення n-того додатку певним виразом. Якщо сума членів ряду наближається до певного скінченого числа, він вважається збіжним. Не усі ряди збігаються. Отже, є певні умови, що визначають збіжність, і в математиці цьому питанню

приділяється багато уваги. При наближеному представленні рядом математичної функції навіть може спостерігатись порушення його збіжності тільки на певній частині області визначення (діапазону значень аргументу), з чим також спробуємо познайомитись в цій роботі.

Завдання (варіант 1)

1. Познайомитись з реалізацією наближеного розрахунку деяких математичних функцій на прикладі експоненти та синуса:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

2. Познайомитись з проблемою збіжності рядів на прикладі:

$$\log(1-x) = -\sum_{n=1}^{\infty} \frac{x^n}{n}$$

Завдання (варіант 2)

1. Познайомитись з реалізацією наближеного розрахунку деяких математичних функцій на прикладі експоненти та косинуса:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$$

2. Познайомитись з проблемою збіжності рядів на прикладі:

$$\log(1+x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n}$$

Хід виконання

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_05 у вказаній викладачем папці.

2. Функція наближеного розрахунку експоненти

2.1. Реалізувати функцію mExp, яка розраховує значення функції e^x на основі ряду Маклорена (повернення результату виконати через return). Обривання ряду виконується, коли поточний доданок стає меншим за значення epsilon, що передається у функцію як аргумент. Функція повинна (через аргументи) повертати кількість доданих членів ряду, яка забезпечила цю умову завершення, та похибку (різницю з бібліотечною функцією exp(x)).

*Реалізацію усіх додаткових функцій робити **після** функції main.*

2.2. **Додаткове:** Для кожної функції у проєкті створити окрему пару файлів - source та header.

2.3. Перевірити працездатність функції викликом з функції main при значеннях epsilon 0.001 та 0,000001 для 11 значень x на проміжку [-5 ; 5]. На консоль не виводити, використовувати налагоджувач!

2.4. Показати результати викладачу.

3. Функція наближеного розрахунку тригонометричної функції

3.1. Аналогічно до п.2.1 ходу виконання реалізувати функцію, яка розраховує значення тригонометричної функції, заданої у п.1 завдання відповідного варіанту. Зрозуміло, що похибка розраховується не відносно експоненти, а до відповідної бібліотечної тригонометричної функції.

3.2. Перевірити працездатність функції викликом з функції main при значеннях epsilon 0.001 та 0,000001 для 9 значень x на проміжку [0; 2π]. На консоль не виводити, використовувати налагоджувач!

3.3. Показати результати викладачу.

4. Аналіз збіжності ряду

4.1. Аналогічно до п.2.1 ходу виконання реалізувати підрахунок ряду, заданого у п.2 завдання відповідного варіанту. У функцію як аргументи передати x та N – максимальне значення n при підсумовуванні (умова виходу з циклу).

4.2. Перевірити працездатність функції викликом з функції main в діапазоні x на проміжку [-1.5; 1.5] з кроком 0.1 . Зверніть увагу на поведінку ряду за межами проміжку]-1; 1[.

4.3. Порівняти результати викликом з функції main при значеннях N 10 та 100 на тому самому проміжку. На консоль не виводити, використовувати налагоджувач!

4.4. Показати результати викладачу.

Отже, якщо завдання виконане в повному обсязі, познайомились з тим, як комп'ютер дозоляє наближеним чином розрахувати певні математичні функції та побачили, що ряди не завжди збігаються. Ну і, звичайно, потренувались використовувати у програмуванні реалізацію та виклик функції.

В цій лабораторній роботі розділення тексту програми на декількох файлів розглядалось як необов'язкове завдання. Прийшов час безпосередньо познайомитись з структуризацією проєкту декількома файлами, що буде зроблено у наступному завданні.

Лабораторна робота №6. Багатофайловий проєкт

Мета роботи: Навчитись розробляти проєкт, текст якого розділений на декілька файлів.

Загальні відомості

При виконанні навчальних проєктів текст виконання одного завдання рідко виходить за сотню рядків програми і включає у себе невелику кількість коротких функцій або пару класів. Такий проєкт на пару–трійку сторінок нескладно читати, навіть коли він зібраний у єдиному файлі.

Реальні прикладні програми розробляються далеко не за пару академічних годин, як завдання лабораторних робіт. Їх обсяг зазвичай виміряється десятками або, навіть, сотнями сторінок. Вони можуть включати сотні функцій або десятки класів. Такий обсяг також можна зібрати в одному спільному файлі – транслятору це заважати не буде. Але програмісту з таким довгим текстом, ще й складним за внутрішньою структурою, працювати незручно – доводиться постійно скролити, звертатись до системи пошуку, активно використовувати закладинки тощо.

При роботі в текстовому редакторі при підготовці великого документу навіть для більш «читабельних», ніж програма, задач, його часто розбивають на окремі частини – як мінімум їх можна відкрити поруч для паралельного перегляду без зайвого пошуку в процесі загальної розробки. Отже, і проєкт розроблюваної програми є сенс розділити на окремі частини. Таке розбиття дає значні переваги, а саме:

- програмісту простіше орієнтуватись в проєкті як, наприклад, в книзі, розділеній на глави;
- при редагування одного з файлів проєкту інші залишаються незмінними, що можна перевірити по їх часових атрибутах – якщо зберігати версії, легко виправити випадкові ушкодження тексту;
- при «швидкій» компіляції файли, в яких не зроблені зміни від попереднього запуску, не транслюються наново – це дещо прискорює компіляцію, до якої при налагодженні доводиться вдаватись навіть при невеликих змінах тексту проєкту;
- до багатофайлогвого проєкту простіше залучити групу програмістів;

- окремий файл чи їх група може розглядатись як своєрідна «текстова» бібліотеки для включення корисних функцій та класів у наступні розробки.

При створенні нового проєкту усі IDE утворюють спеціальний службовий «файл проєкту», що його конфігурує. На жаль формати такого файлу не сумісні між різними IDE. Серед іншого в проєкті фіксується, які файли враховуються при компіляції – просто внести файл ззовні у відповідну папку недостатньо.

Розділення загального тексту на частини є неоднозначною задачею, але є певні рекомендації, сформовані на багаторічному досвіді багатьох людей. При використанні процедурного програмування, тобто на основі використання функцій (без класів) для окремої функції або декількох змістовно пов'язаних створюють два файли (розглядаємо C / C++, при роботі з іншими мовами дещо інакше). Обом дають однакове ім'я (для зручності таке, що відповідає назві функції), але з рідними розширеннями:

- **.**c або **.**cpp відповідно для C та C++ – це файл «джерела» (англ. source) коду;
- **.**h – заголовочний (англ. header).

У файлі джерела реалізують саму функцію, включаючи в нього її рядок опису та тіло (виконувану частину). Заголовочний файл містить її прототип (згадаємо, що це таке і його синтаксис). Він через дії препроцесора (директива #include) вноситься в інші файли, в яких ця функція використовується. Зауважимо, що на відміну від посилання для включення на файли стандартної бібліотеки назву заголовочного файлу з проєкту включають не в кутові дужки, а у лапки, що означає «шукати в папці проєкту» (реально – трохи складніше, але це опустимо). З врахуванням того, що зазначали раніше, файли (новостворені або готові зовнішні) до проєкту слід додавати тільки через відповідні команди IDE.

Завдання

Розробити проєкт, що включає декілька файлів з функціями, які використовуються сумісно. В даній роботі різниця в завданнях варіантів є значною і винесена безпосередньо в «хід виконання» – саме в ньому містяться два варіанти лабораторної.

Хід виконання (варіант 1)

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_06 у вказаній викладачем папці.

1.2. Додати до проєкту три пари (source + header) файлів. У кожній парі назви відповідають подальшим завданням:

- “operations” (операції);
- “functions” (функції);
- “series” (ряди).

2. Функції арифметичних операцій

2.1. Реалізувати у відповідному souce-файлі (має розширення імені cpp) функції pls, mns, mult, div, які виконують арифметичні дії з дійсними аргументами, результат повертається через return. Прототипи внести у header-файл (має розширення імені h).

2.2. Перевірити працездатність функцій викликом з функції main.

3. Функції розрахунку математичних функцій

3.1. Реалізувати у відповідному source-файлі (має розширення імені cpp) функції pow_m, div_pow, які виконують розрахунок $y = x^n$ та $y = 1/x^n$ (ступінь є цілим додатним значенням!). Результат повертати через return.

3.2. Зведення у ступінь повинно бути реалізоване множенням.

3.3. Усі арифметичні дії виконувати викликом раніше реалізованих функцій арифметичних операцій (п.2).

Зауваження: для забезпечення доступу до попередньо реалізованих функцій у файл, де використовується їх виклик, треба включити (#include) відповідний header-файл (з прототипами)

3.4. Перевірити працездатність функцій викликом з функції main.

4. Функції розрахунку рядів

4.1. Реалізувати у відповідному source-файлі (має розширення імені cpp) функції s1, s2, які розраховують ряди $s1 = \sum_{nMin}^{nMax} n^k$ та $s1 = \sum_{nMin}^{nMax} 1/n^k$

4.2. Усі арифметичні дії виконувати викликом раніше реалізованих функцій арифметичних операцій (п.2). Для зведення у ступінь використовується розробка математичних функцій (п.3).

4.3. Перевірити працездатність функцій викликом з функції main.

4.4. Продемонструвати проєкт викладачу.

Хід виконання (варіант 2)

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_06 у вказаній викладачем папці.

1.2. Описати за зразком (рис.6.1) дані у файлі main.cpp.

```
int A[] ={12,5,34,6,15,14,100, 33, 25, 16};
int B[] ={2, 15,3,26,115,1,123, 31, 5, 16};
int C[] ={2, 3, 5, 11};
int len1=10;
int len2=4;
```

Рис.6.1

1.3. Додати до проєкту три пари (source + header) файлів. У кожній парі назви відповідають подальшим завданням:

- “oper” (операції);
- “fn” (функції);
- “count” (підрахунки).

2. Функції арифметичних операцій

2.1. Реалізувати у відповідному source-файлі (має розширення імені cpp) функції iPlus, iMinus, iMult, iDiv, iOst, які виконують арифметичні дії та розрахунок остачі від ділення для цілочисельних аргументів, результат повертається через return. Прототипи внести у header-файл (має розширення імені h).

2.2. Перевірити працездатність функцій викликом з функції main.

3. Функції перевірки властивостей аргументів

3.1. Реалізувати у відповідному source-файлі (має розширення імені cpp) функції plusMinusDividedTo, multDividedTo, які виконують перевірку – чи ділиться націло на третій аргумент сума або різниця перших двох аргументів (перша з функцій), або результат множення перших двох аргументів (друга з функцій). Результат повертати через return.

3.2. Усі арифметичні дії виконувати викликом раніше реалізованих функцій арифметичних операцій (п.2).

Зауваження: для забезпечення доступу до попередньо реалізованих функцій у файл, де використовується їх виклик, треба включити (#include) відповідний header-файл (з прототипами)

3.3. Перевірити працездатність функцій викликом з функції main.

4. Функції додаткових розрахунків

4.1. Реалізувати у відповідному source-файлі (має розширення імені cpp) функцію count1, яка виконує підрахунок – скільки пар чисел з однаковим індексом з двох глобальних масивів A, B утворюють суму або

різницю, що націло ділиться на хоча б один елемент з масиву C. Дані у функцію передавати як глобальні змінні, повернення результату через return.

4.2. Усі арифметичні дії та перевірку кратності виконувати викликом раніше реалізованих функцій арифметичних операцій (п.2) та властивостей аргументів (п.3).

4.3. Реалізувати у тому ж source-файлі функції count2, , яка виконує підрахунок – скільки пар чисел з однаковим індексом з двох глобальних масивів A, B утворюють добуток, що націло ділиться на хоча б один елемент з масиву C. Дані у функцію передавати як глобальні змінні, повернення результату через return.

Отже, тепер за потреби маємо можливість структурувати проєкт не тільки правильним розбиттям на рядки та використанням функцій, а й більш глобальним чином – розділенням на декілька файлів. При бажанні цю нову навичку можна вдосконалювати і в наступних проєктах.

Лабораторна робота №7. Масиви

Мета роботи: Навчитись обробляти дані одно- та двовимірних масивів.

Загальні відомості

При розв'язанні багатьох задач, що вимагають використання програмування, виникає потреба виконувати однакову за алгоритмом генерацію або обробку великої кількості даних. Така повторювана дія може бути реалізована циклом, але в його тілі виникає потреба своєрідної уніфікації звернення до даних – в кожній ітерації (окремому проходженні тіла) циклу треба звертатись до іншого елемента даних, але «майже однаковим» чином. Отже, класичний варіант набору змінних не підходить, бо різними мають бути їхні імена.

Для вирішення описаної проблеми структуризації даних в програмуванні використовується масив – впорядкований набір певної кількості елементів, звернення до яких виконується через спільне ім'я та ключ, що виділяє один конкретний елемент з усіх в масиві. Класичні (і серед них обрана нами) мови програмування спираються виключно на використання в якості ключа номера (індексу). Більш сучасні додатково підтримують асоціативні масиви, в яких ключ є рядком (текстом).

Оскільки ми використовуємо C / C++, будемо виходити з того, що елементи масиву є обов'язково однотипними, їх тип задається при його описі (тоді ж задається ім'я та кількість елементів) і надалі не може бути змінений – згадуємо про статичну типізацію. Завдяки таким обмеженням використовується послідовне розташування елементів в оперативній пам'яті, що разом з фіксованою їх кількістю забезпечує максимально швидкий доступ процесора до кожного з елементів, а висока швидкість виконання програми є однією з принципових переваг цієї мови.

Слід пам'ятати, що нумерація елементів починається з 0, а останнім індексом елемента масиву довжини N, до якого можна звертатись без помилки, є N-1. І ще одна важлива особливість та ціна забезпечення максимальної швидкості обробки в обраній нами мові – відстеження такої коректності меж масиву (тобто НЕ вихід за його границю) покладається на програміста.

Фіксована від моменту його створення довжина масиву призводить до певної незручності. Якщо його описувати звичайним чином, як класичну змінну (тобто статично), вже при

розробці програми потрібно враховувати, яка реальна довжина даних буде потрібна при її виконанні, а в багатьох випадках від запуску до запуску з різними даними вимога до кількості елементів може значною мірою змінюватись.

Саме тому найбільш правильним використанням масивів (за винятком деяких простих випадків) є динамічне їх створення з використанням механізму вказівників. При цьому як опис масиву використовується опис вказівника на відповідний тип, а сам масив створюється дією оператора new (пам'ятаємо, що на кожен «new» повинен бути «delete»). Завдяки цьому саме при виконанні програми можна визначити (з завантаженого завдання або розрахунків) потрібну кількість елементів, що і використати при виділенні пам'яті.

Динамічне створення двовимірного масиву має додаткову специфіку – маємо справу з масивом, елементами якого є масиви. Отже, спочатку (через new) необхідно створити масив вказівників на рядки (тип – вказівник на тип елемента масиву). Надалі індекс цього масиву задає саме номер рядка. Потім проходженням в циклі створюємо (також через new) самі рядки (одновимірні масиви необхідної довжини), посилання на які записується в елементи зазначеного раніше масиву рядків. Видалення відбувається у протилежному порядку – спочатку в циклі видаляються рядки, а потім масив рядків.

Передача даних у функцію також має свої особливості в порівнянні зі звичайними змінними. Незалежно від того, статично чи динамічно створений масив, у функцію передається посилання на його початок (про цю особливість в посібнику вказували раніше), отже немає способів всередині функції визначити кількість елементів. Через це необхідно цю кількість передавати у функцію разом з масивом. Є також певні тонкощі з передачею двовимірних масивів. В цьому плані можна порекомендувати в якості відповідного формального параметра при описі функції використовувати вказівник на вказівник (дві зірочки після типу в описі параметра).

Завдання

Розробити функції, що виконують створення масивів, внесення в їхні елементи даних та певну обробку. В даній роботі різниця в завданнях варіантів є значною і винесена безпосередньо в «хід виконання» – саме в ньому містяться два варіанти лабораторної.

Хід виконання (варіант 1)

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_07 у вказаній викладачем папці.

1.2. Додати до проєкту та налагодити функцію виведення на консоль одновимірного масиву (рис.7.1).

```
void print(double* a, int N)
{
    int n=0;
    while (n<N) printf("%5.3g   ", a[n++]);
    printf("\n-----\n");
}
```

Рис.7.1

2. Функція ініціалізації одновимірного масиву

2.1. Реалізувати функцію, яка за заданим N створює (динамічно!) і повертає одновимірний масив довжиною N. та заповнює його косинусом відстані від останнього елемента.

2.2. Перевірити працездатність функції викликом з функції main та виведенням масиву на консоль.

3. Функція ініціалізації двовимірного масиву

3.1. Реалізувати функцію, яка створює (динамічно!) і повертає двовимірний масив та заповнює його елементи значеннями декартової відстані від елемента з найбільшими індексами.

При розробці проєкту можна використовувати (за вибором студента) двовимірні масиви у форматі a[i][j] або їх зведення до одновимірного!

3.2. Реалізувати функцію виведення на консоль двовимірного масиву аналогічно до виведення на консоль одновимірного масиву (п.1.2).

3.3. Перевірити працездатність функції (п.3.2) викликом з функції main з виведенням масиву (п.3.1) на консоль.

4. Функція ініціалізації квадратного масиву

4.1. Реалізувати функцію, яка створює (динамічно!) і повертає квадратний двовимірний масив та заповнює його елементи значеннями 10*i+j .

4.2. Перевірити працездатність функції викликом з функції main та виведенням масиву на консоль.

5. Функція обробки даних із масивів

5.1. Реалізувати функцію, яка виконує транспонування масиву та виводить результат через return.

5.2. Реалізувати функцію, яка виконує поелементне усереднення двох масивів і виводить результат через return.

5.3. Реалізувати функцію з тим же іменем, що у п.5.2, яка виконує поелементне усереднення двох масивів, але виводить результат через аргумент функції.

5.4. Перевірити працездатність функцій викликом з функції main та виведенням результатів на консоль.

Хід виконання (варіант 2)

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_07 у вказаній викладачем папці.

1.2. Додати до проєкту та налагодити функцію виведення на консоль одновимірного масиву (рис.7.2).

2. Функція ініціалізації одновимірного масиву

2.1. Реалізувати функцію, яка за заданим N створює (динамічно!) і повертає одновимірний масив довжиною 2N+1 та заповнює його таким чином:

- "центральний елемент" a[N] дорівнює 2;
- інші елементи дорівнюють оберненій (1/x) відстані від "центрального" елемента a[N].

2.2. Перевірити працездатність функції викликом з функції main та виведенням масиву на консоль.

```
void print(double* a, int N)
{
    int n=0;
    while (n<N) printf("%5.3g   ", a[n++]);
    printf("\n-----\n");
}
```

Рис.7.2

3. *Функція ініціалізації двовимірного масиву*

3.1. Реалізувати функцію, яка створює (динамічно!) і повертає двовимірний масив та заповнює його елементи значеннями декартової відстані від елемента з мінімальними індексами.

При розробці проєкту можна використовувати (за вибором студента) двовимірні масиви у форматі a[i][j] або їх зведення до одновимірного!

3.2. Реалізувати функцію виведення на консоль двовимірного масиву аналогічно до виведення на консоль одновимірного масиву (п.1.2).

3.3. Перевірити працездатність функції (п.3.2) викликом з функції main з виведення масиву (п.3.1) на консоль.

4. *Функція ініціалізації квадратного масиву*

4.1. Реалізувати функцію, яка створює (динамічно!) і повертає квадратний двовимірний масив та заповнює його елементи значеннями $(i^2 + j^2)/(1+j)$.

4.2. Перевірити працездатність функції викликом з функції main та виведенням масиву на консоль.

5. Функція обробки даних із масивів

5.1. Реалізувати функцію, яка виконує поелементне додавання до квадратного масиву другого масиву, але транспонованого. Результат виводити через return.

5.2. Перевірити працездатність функції викликом з функції main та виведенням результатів на консоль.

6. Функції коректного видалення динамічних масивів

6.1. Реалізувати функцію, яка коректно видаляє виділений оператором new блок даних одновимірного масиву. При цьому повинна блокуватись спроба повторного видалення.

6.2. Перевірити працездатність функцій викликом з функції main.

6.3. Реалізувати функцію, яка коректно видаляє виділений оператором new блок даних двовимірного масиву. При цьому повинна блокуватись спроба повторного видалення.

6.4. Перевірити працездатність функцій викликом з функції main.

Зазначимо, що у виконаному завданні (якщо не брати до уваги початкового) вперше була знята заборона на консольний вивід – бо при роботі з масивом, особливо двовимірним необхідно бачити надто багато даних разом. І на жаль, при відтворені через посилання (вказівник) далеко не усі налагоджувачі (в тому числі у Code::Blocks) в інструменті перегляду змінних дозволяють розкривати у зручній формі вміст масиву. Через це при налагоджуванні в самій функції, куди передається масив, краще за все статично описати тимчасовий масив з тестовими даними. Отже, розібрались з тим, як використовувати масиви на абстрактних прикладах. Тепер прийшов час спробувати їх використати для побудови конкретного обчислювального інструменту, який є досить популярним в математиці.

Лабораторна робота №8. Масиви +

Мета роботи: Навчитись використовувати двовимірні масиви в практичних задачах (на прикладі розрахунку детермінанту).

Загальні відомості

Перед виконанням програмного проєкту цієї роботи згадаємо, що таке визначник або детермінант матриці. Зазначимо, що матриця, отже і детермінант, як її характеристика, є досить популярними в математиці та галузях, які її використовують.

Отже, матрицею називається прямокутна таблиця даних (нас для розрахункової обробки цікавить матриця чисел), упорядкована за рядками та стовпцями. Така структуризація даних спрощує опис та реалізацію багатьох операцій з блоками чисел, що зробило її основним інструментом лінійної алгебри, а через неї і аналітичної геометрії, яка використовується, наприклад, при моделюванні будь-якої системи, що реалізується у просторі. При цьому використовують такі операції, як додавання матриць однакового розміру, множення матриці на матрицю, транспонування тощо. А в багатьох математичних задачах, наприклад для розв'язання системи лінійних рівнянь або в задачах на власні числа, використовують розрахунок визначника квадратної матриці (вона може бути і не квадратною, але для визначника є таке обмеження).

Визначником матриці, яка складається з чисел, є число, яке обраховується за участі усіх її елементів. Його значення може бути розраховане декількома еквівалентними способами. Одним з них є формула Лейбніца – сума добутків елементів матриці які будуються таким чином, що в кожному з них є по одному елементу з кожного рядка і кожного стовпця, а знак цього добутку визначається парністю перестановки номерів. В принципі, реалізація такого розрахунку можлива через цикли та логічні вирази, але якщо спробуєте це зробити на основі початкового досвіду програмування, скоріше за все нічого не вийде. Тому краще використати інший алгоритм – розклад Лапласа як лінійну комбінацію визначників порядку, меншого на одиницю. Менші визначники формуються викреслюванням рядка, по якому розкладаємо, та стовпчика, що відповідає елементу цього рядка, множенням на який зазначеного субвизначника формується кожний доданок. Знак доданку

визначається парністю його номера в рядку (починаємо з нуля, непарні від'ємні). Саме формування субвизначників і буде найскладнішою складовою даного проєкту.

При виконанні розрахунків в комп'ютерній програмі матриця реалізується як двовимірний масив. Отже, для використання її в реальному програмному проєкті в попередніх лабораторних були відпрацьовані всі потрібні дії. Залишається трохи потренуватись на цьому, дещо складнішому, завданні.

Завдання

Реалізувати функції розрахунку детермінантів матриць розмірами 2×2, 3×3, 4×4. Завдання цієї лабораторної не має окремих варіантів.

Хід виконання

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_08 у вказаній викладачем папці.

1.2. У функції main описати три дійсних двовимірних масиви з контрольними даними (рис.8.1).

```
double A[][2]=  {
     {4,8},
     {3,7}        };

double B[][3]=  {
     {1,4,8},
     {2,5,9},
     {3,7,10}     };

double C[][4]=  {
     {1,5,10,14},
     {2,6,11,16},
     {3,8,12,17},
     {4,9,13,20} };
```

Рис.8.1

2. Функція розрахунку детермінанта матриці 2×2

2.1. Реалізувати функцію det2, яка розраховує значення детермінанта матриці 2×2 і повертає результат через

return. При описі функції тип аргументу «двовимірний масив» задається як double a [ ] [2].

2.2. Перевірити працездатність функції викликом з функції main. Правильність розрахунків можна перевірити використанням Excel або аналогу.

3. Функція розрахунку детермінанта матриці 3×3

3.1. Реалізувати функцію det3, яка розраховує значення детермінанта матриці 3×3 через зведення до детермінантів 2×2, а далі використати результат виконання п.2. Результат, як і раніше, повернути через return.

3.2. Перевірити працездатність функції викликом з функції main – і знову Excel.

4. Функція розрахунку детермінанта матриці 4×4

4.1. Реалізувати функцію det4, яка розраховує значення детермінанта матриці 4x4 через зведення до детермінантів 3x3 і повертає результат через return. При цьому використати виклик функції п.3.

4.2. Перевірити працездатність функції викликом з функції main.

При виконанні даного завдання можна побачити, що det4 фактично є деякою модифікацією det3. Отже, за необхідності процес розробки даної лінійки функцій можна продовжити на випадки матриць більшого розміру – копіпаст і невеличкі досить прості виправлення. Але…

Є кращі способи. Згадаємо про рекурсивну функцію (на лекціях був приклад з факторіалом). Аналогічно до зробленого завдання можна написати універсальну функцію, в яку передається матриця довільного розміру, але вона використовує при розкладі по рядку для розрахунку менших детермінантів виклик не іншої подібної функції, а себе. А ще можна розрахувати визначник як добуток діагональних елементів після зведення матриці до форми верхнього трикутника відповідним перетворенням рядків. При цьому функцію також можна зробити універсальною щодо розмірів.

Лабораторна робота №9. Файли та консоль

Мета роботи: Навчитись використовувати ввід та вивід даних програмного проєкту.

Загальні відомості

Прикладна програма використовує для своєї роботи дані від користувача, різні від запуску до запуску. Зазначимо, що значна частина сучасних програм є інтерактивними. Такі програми мають постійну взаємодію з користувачем для отримання нової частини завдання (як, наприклад при введенні та редагуванні тексту в редакторі). Проте є й інші програми, які завантажують вхідні дані пакетом для подальшої обробки. Цей варіант роботи програми має довгу історію використання, але зберігає актуальність і в сучасних умовах. Він використовується, наприклад, при комп'ютерному моделюванні. Виведення даних в цьому випадку виконується або в процесі розрахунків, або наприкінці, але як і вхідні дані, вони мають вигляд взаємозв'язаного блоку.

Зазвичай дані, що вводяться, не зводяться до двох – трьох значень. Отже, їхнє введення з клавіатури має суттєві недоліки, до речі, як і виведення на консоль:

- в разі помилки оператора при введенні даних доводиться перезапустити програму і все повторити з початку;
- часто блоки даних, які вводяться при різних запусках, між собою мають багато спільного і усе це кожного разу треба вводити наново;
- виведення у консольне вікно не зберігає дані після його закриття.

Саме тому правильним режимом вводу – виводу є використання файлів. Найпростіше використовувати для цього файли текстового формату, оскільки в цьому випадку вхідні дані можна готувати до завантаження у довільному текстовому редакторі. Так само, результати розрахунків видно у редакторі і до них легше адаптувати програму для подальшої обробки, наприклад для накопичення даних або побудови графіків. При цьому можна звернутись і до подальшої роботи в табличних процесорах з офісних пакетів, які непогано імпортують структуровані у вигляді колонок дані зі звичайних текстових файлів.

Завдання (варіант 1)

1. Реалізувати початкову структуру проєкту за зразком на рис.9.1.

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

using namespace std;

double F( /* . . . */ )
{
}
void ConsoleOut( /* . . . */ )
{
}
void ExcelOut( /* . . . */ )
{
}
void read( /* . . . */ )
{
}
void readFName( /* . . . */ )
{
}
int main()
{
    float A=2;
    float omega1=1;
    float phi1=0.5;
    float omega2=1.5;
    float t1=0;
    float t2=12;
    float dt=0.25;
    char fname[20];
    strcpy(fname, "param_1.txt");

    readFName( /* . . . */ );
    read( /* . . . */ );
    ConsoleOut( /* . . . */ );
    return 0;
}
```

Рис.9.1

2. Створити (за допомогою блокноту Windows) два текстових файли з різними наборами даних для введення у програму за прикладом на рис.9.2.

```
A= 3
omega1= 1.5
phi1= 0.8
omega2= 4.5
t1= 1
t2= 8
dt= 0.1
```

Рис.9.2

3. Реалізувати виведення програмою таблиці з даними функції $y = A\sin(\omega_1 t + \varphi_1) + \cos(\omega_2 t)$ в проміжку від t_1 до t_2 з кроком dt на основі даних, що вводяться з раніше створеного вхідного файлу. Для цього використати виведення двох колонок у вихідні дані.

Завдання (варіант 2)

1. Реалізувати початкову структуру проєкту за зразком на рис.9.3 (а,б).

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

using namespace std;
double F( /* . . . */ )
{
}
void consoleOut( /* . . . */ )
{
}
void excelOut( /* . . . */ )
{
}
void parameters( /* . . . */ )
{
}
void readFName( /* . . . */ )
{
}
```

Рис.9.3 – а

```
int main()
{
    float a=2;
    float b1=1;
    float b2=-2;
    float c=0.5;
    float d=1.5;
    float dd=0.5;

    float x1=0;
    float x2=5;
    float dx=0.25;

    char fname[20];
    strcpy(fname, "param_1.txt");

    readFName( /* . . . */ );
    parameters( /* . . . */ );
    consoleOut( /* . . . */ );
    return 0;
}
```

Рис.9.3 – б

2. Створити (за допомогою блокноту Windows) два текстових файли з різними наборами даних для введення у програму за прикладом на рис.9.4.

```
a= 1
b1= -32
b2= 24
c= 50
d= 5
x1= -2
x2= 2
dx= 0.25
```

Рис.9.4

3. Реалізувати виведення програмою таблиці з даними функції $y = ax^3 + bx^2 + cx + d$ при двох значеннях b (дві криві разом) в проміжку від x_1 до x_2 з кроком dx на основі даних, що вводяться з раніше створеного вхідного файлу. Для цього використати виведення трьох колонок вихідних даних.

Хід виконання

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_09 у вказаній викладачем папці.

1.2. Реалізувати початковий стан проєкту відповідно до п.1 в своєму варіанті завдання.

2. Читання вхідних даних з файлу

2.1. Сформувати два текстових файли з вхідними даними param_1.txt та param_2.txt зі структурою рядків відповідно до зразку у завданні.

2.2. Реалізувати функцію, яка буде отримувати через введення з клавіатури ім’я файлу з вхідними даними, який потрібно прочитати.

2.3. Реалізувати введення вхідних даних з файлу функцією, реалізованою у п.2.2..

3. Виведення на консоль

3.1. Реалізувати виведення на консоль таблиці з даними функції заданої в п.3 в своєму варіанті завдання (краще використовувати функції виведення мови C, а не C++).

3.2. Перевірити працездатність, показати викладачу, в тому числі з вибором файлу з вхідними даними.

4. Виведення у файл для імпорту в Excel

4.1. Реалізувати виведення таблиці з даними функції, заданої в п.3 в своєму варіанті завдання у файл формату, прийнятного для імпорту в Excel або аналог.

4.2. Перевірити, показати результати викладачу.

Отже, вперше використали консоль «на повну». – саме запит імен файлів для вводу та виводу слід залишити для використання клавіатури. Але пам’ятаємо про те, що консольний застосунок як обробник може взагалі отримувати дані від оболонки, реалізованої з графічним інтерфейсом користувача і це найпростіше також робити через файли.

Лабораторна робота №10. Випадкова величина

Мета роботи: Дослідити якість генерування випадкової величини.

Загальні відомості

Отже, основні ідеї використання процедурного програмування те його практичну реалізацію однією з найбільш популярних мов (C / C++) розібрали. Тепер спробуємо додатково потренуватись. Але для цього краще обрати не щось надзвичайно абстрактне, а те, що активно використовується саме в прикладних програмах, в першу чергу для моделювання, оскільки воно є надзвичайної важливим в сучасній інженерії.

При моделюванні досліджується поведінка певного об’єкту або системи об’єктів в залежності від додаткових умов. У випадку, коли за поточним станом системи можна однозначно визначити наступні, процес називається детермінованим. Формалізація багатьох законів природи призводить до формул, які і задають таку поведінку. Однак в реальних ситуаціях додаються певні непередбачувані впливи, які цю детермінованість дещо порушують. Процес, майбутній стан якого неможливо передбачити, називається випадковим. Як приклад, можна навести флуктуації температури, шумову складову електричного або акустичного сигналу, зміну вітру, яка впливає на траєкторію польоту дронів, і багато іншого. Таку випадковість поведінки досліджуваного об’єкта у багатьох випадках необхідно реалізувати при моделюванні. А частіше за все реалізується деяка комбінація детермінованого та випадкового процесів.

Хоча у випадковому процесі не можна забезпечити точне передбачення, зазвичай можна визначити розвиток з певною ймовірністю. Наприклад, коли ви кидаєте гральний кубик, якщо він не шулерський, при дуже великій кількості спроб кожна грань буде випадати майже однакову кількість разів. І чим більшою буде кількість спроб тим меншим буде відхилення від однакового значення. Виконанням даної лабораторної роботи спробуємо у цьому пересвідчитись. При аналізі випадкових процесів використовують розрахунок певних характеристик на основі обробки деякої кількості незалежних реалізації. Найважливішими є середнє значення та дисперсія (хоча є ще багато й інших цікавих показників), що й буде використано.

При реалізації випадкового процесу в програмі є потреба у функції, яка буде повертати випадкову величину – кожного разу інше значення, але з розподілом його ймовірності, що буде відповідати певному закону. Такай закон частіше за все описується деякою формулою. Найчастіше використовують рівномірний (ймовірність для усіх значень однакова) та нормальний (описується функцією Гаусса). Сучасні мови програмування мають функцію стандартної бібліотеки, яка є генератором дискретної псевдовипадкової величини з рівномірним розподілом. Крім того, за необхідності мати послідовність значень з нормальним розподілом можна використати відомі математичні алгоритми, які дозволяють їх розрахувати на основі значень рівномірного.

Декілька слів про приставку «псевдо». Якщо викликати зазначену раніше функцію в циклі багато разів, дійсно отримаємо рівномірність розподілу значень, але при кожному запуску програми сама послідовність буде однаковою. До речі, при налагоджуванні ця особливість є навіть корисною, бо декілька послідовних значень швидко запам’ятовуються. Але реальний запуск вимагає саме випадковості, що може бути забезпечено випадковим вибором початкової точки послідовності. А комп’ютер має пристрій, який може забезпечити справжню випадковість у виборі номера початкового елемента. Це годинник, з якого просто зчитується час у певний момент після запуску програми.

Завдання

1. Реалізувати обчислювальний експеримент на основі випадкової величини.
2. Дослідити як кількість реалізацій випадкової величини впливає на правильність розподілу.

Завдання цієї лабораторної не має окремих варіантів.

Хід виконання

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_10 у вказаній викладачем папці.

1.2. Реалізувати початкову структуру проєкту за зразком на рис.10.1.

```
#include <iostream>
#include <cstdlib>
#include <stdio.h>
#include <math.h>
#include <time.h>

const int dimension = 6;              //1000;
int Nrealization = 1000;              //   = 1e6;

double P[dimension];                  //probability
double Pmid;                          //mid value of P;

//----------------------------
void init(double P[], int len, int Nrealization)
{
//----------------------------
double countSkv(double P[], int len)
{

int main()
{
```

Рис.10.1

2. Розрахунок масиву ймовірностей реалізації випадкової величини

2.1. Дослідити дію функції rand

```
int realization = rand();
```

з попереднім (одноразовим!) викликом

```
srand(time(NULL));
```

та без нього.

2.2. Реалізувати функцію init, яка заповнює елементи масиву P ймовірністю того, що випадкова величина має значення, що відповідає номеру елемента масиву.

2.3. Перевірити налагоджувачем працездатність функції

3. Розрахунок середньоквадратичного відхилення

3.1. Реалізувати функцію countSkv, у якій спочатку розраховується середнє значення усіх елементів масиву

$$\mathbf{X}_{\text{mid}} = 1/N \sum_{n=0}^{N-1} X_n$$

а потім (в тій самій функції, як продовження) з його використанням розрахується СКВ

$$СКВ = \sqrt{1/N \sum_{n=0}^{N-1} (X_n - X_{mid})^2}$$

3.2. Перевірити налагоджувачем працездатність функції.

4. Відношення СКВ до середнього значення величини в залежності від кількості реалізацій

4.1. У функції main реалізувати виведення на консоль в табличній формі досліджуваної залежності (перша колонка – кількість реалізацій, друга СКВ/Pmid).

4.2. Повторити дослідження для кількості значень випадкової величини (dimension) 100.

4.3. Показати результати викладачу.

Сподіваємось, перша спроба реалізувати власний проєкт із залученням обчислювальної математики була успішною. Розв’язана задача є дуже важливою. На основі використання випадкової послідовності та статистичної обробки отриманих результатів навіть розроблена та досить активно застосовується група чисельних методів із загальною назвою – метод Монте-Карло.

Лабораторна робота №11. Чисельне інтегрування

Мета роботи: Дослідити збіжність реалізації чисельного інтегрування.

Загальні відомості

Слід пам'ятати про те, що не зважаючи на прагнення до максимальної абстрактності математичні розробки в більшості своїй мають чіткий практичний зміст і використовуються для розв'язання прикладних задач.

Однією з важливих розділів математичного аналізу є інтегральне числення. Операція інтегрування є протилежною до диференціювання і зводиться до пошуку первісної, тобто такої функції, диференціювання якої утворює підінтегральну. Одним з важливих практичних застосувань інтегрування є пошук площі або об'єму. Для цього використовується визначений інтеграл, що розраховується як різниця значень первісної при підстановці в неї границь інтегрування. Нібито просто – одне віднімання. Але усе так красиво тільки у випадках, коли вдається знайти аналітичний вигляд первісної, а це на жаль далеко не завжди можливо. Як варіант – трохи змінити підінтегральну функцію, щоб досягти аналітики. Але цим вноситься похибка, іноді неконтрольована.

Альтернативою є використання наближеного розрахунку з «правильною» підінтегральною функцією. Для цього інтервал інтегрування розбивається на певну кількість кроків – чим більша їх кількість, тим меншою буде похибка наближеного обчислення. Повну площу можна розрахувати як суму частин. Поки що похибку цим не вносимо, але навіть на маленькому інтервалі первісна невідома. І ось тут виникає можливість більш ефективно використати наближення. Невідому первісну на зазначеному маленькому інтервалі апроксимують деякою простішою функцією, яку легко обчислювати – прямокутником, трапецією, поліномом невеликої степені. Є й інші методи наближеного інтегрування, але обмежимось саме цими.

Завдання

1. Реалізувати чисельний розрахунок визначеного інтегралу трьома методами.
2. Порівняти швидкість збіжності реалізованих методів.

Завдання цієї лабораторної не має окремих варіантів.

Хід виконання

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_11 у вказаній викладачем папці.

1.2. Реалізувати початкову структуру проєкту за зразком на рис.11.1.

1.3. Реалізувати тестову функцію F, яка повертає значення деякої математичної функції від значення x, яке передається у функцію як параметр

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

double F(double x)
{
}

double rect(double Xmin, double Xmax, int N, double (*F)(double))
{
}

double trap(double Xmin, double Xmax, int N, double (*F)(double))
{
}

double simp(double Xmin, double Xmax, int N, double (*F)(double))
{
}

int main()
{
}
```

Рис.11.1

2. Функція розрахунку визначеного інтегралу методом прямокутників

2.1. Реалізувати функцію rect, яка повертає значення визначеного інтегралу, що розраховується як сума площ прямокутників (рис.11.2) в межах від Xmin до Xmax з розбиттям цього проміжку на N кроків.

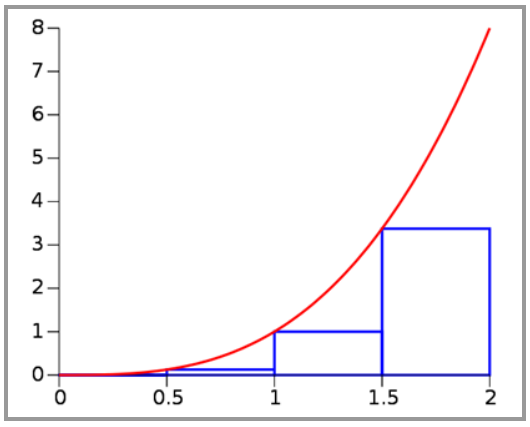


Рис.11.2

2.2. Перевірити налагоджувачем працездатність функції.

3. Функція розрахунку визначеного інтегралу методом трапецій

3.1. Реалізувати функцію trap, яка повертає значення визначеного інтегралу, що розраховується як сума площ трапецій (рис.11.3) в межах від Xmin до Xmax з розбиттям цього проміжку на N кроків.

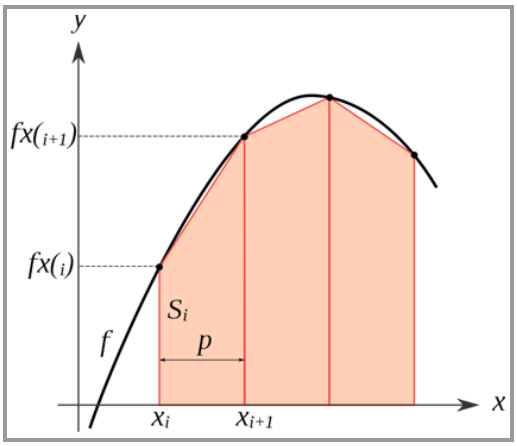


Рис.11.3

3.2. Перевірити налагоджувачем працездатність функції.

4. Функція розрахунку визначеного інтегралу методом Сімпсона

4.1. Реалізувати функцію simp, яка повертає значення визначеного інтегралу, що розраховується як сума площ криволінійних трапецій (рис.11.4) в межах від Xmin до Xmax з розбиттям цього проміжку на N кроків

$$\int_a^b f(x)\,dx \approx \frac{\Delta x}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 4f(x_{n-1}) + f(x_n)\right).$$

4.1. Перевірити налагоджувачем працездатність функції.

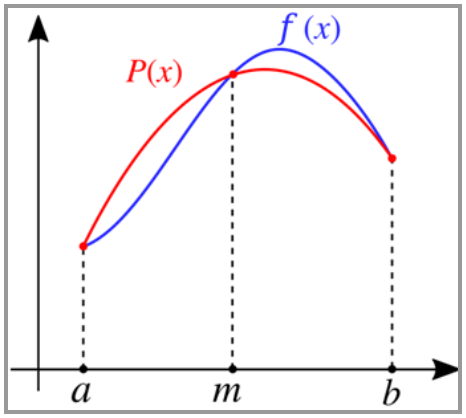


Рис.11.4

5. Порівняння збіжності інтегрування трьома реалізованими методами

5.1. У функції main вивести на консоль залежність значень інтегралу трьома методами від кількості кроків при інтегруванні функції $y = x^2$ на проміжку від 3 до 5.

5.2. У функції main вивести на консоль залежність значень інтегралу трьома методами від кількості кроків при інтегруванні функції $y = \sin^2 5x$ на проміжку від 3 до 5.

5.3. Показати результати викладачу.

Отже, познайомились з одним з випадків, коли наближені обчислення на комп'ютері, забезпечуючи «контрольованість» похибки, мають перевагу перед аналітикою. В наступній задачі розглянемо ще один подібний випадок.

Лабораторна робота №12. Диференціальне рівняння

Мета роботи: Познайомитись із чисельним розв'язанням диференціального рівняння з початковими умовами (задача Коші).

Загальні відомості

Диференціальними називаються рівняння, які зв'язують певні функції з їхніми похідними. Вони з'являються при математичному описі багатьох практичних задач, зазвичай за рахунок використання формального опису відповідних законів природи. Через це такі рівняння є основою математичного моделювання у багатьох наукових галузях – фізиці, хімії, біології тощо. З врахуванням того що забезпечення точності моделювання в інженерних задачах спирається на адекватність врахування фізичних законів, в інженерії вони також супроводжують майже будь-яке моделювання.

Як і при розв'язанні алгебраїчних рівнянь, коли шукають певне число (або числа) які вдовольняють рівнянню при їх підстановці, розв'язання диференціального рівняння зводиться до пошуку певного об'єкту, який вдовольняє рівнянню, але цим об'єктом є не число, а функція.

Математика багато уваги приділяє методам розв'язання таких рівнянь, але тільки відносно прості з них мають розв'язок, що задається формулою. І, навіть, невеликі ускладнення функцій, що є частиною рівняння, можуть порушити таку можливість. Як приклад, можна навести рівняння коливань осцилятора, в якому для отримання розв'язку синус доводиться апроксимувати аргументом, що вже є джерелом похибки та обмежує правильність розв'язку (прийнятний рівень похибки) «малими» відхиленнями від стаціонарного стану системи. Отже, маємо «точний» розв'язок «неточної» задачі.

Чисельні методи дозволяють зробити навпаки – отримати «неточний», тобто наближений, розв'язок, але «точного» рівняння. І при використанні цих методів є можливість жертвуючи збільшенням загального часу розрахунків (кількості елементарних дій), зменшити похибку ланцюжка обчислень. Як і при інтегруванні в попередній роботі, коли для цього робили розбиття інтервалу інтегрування на більшу кількість дрібніших кроків. Отже, в багатьох випадках маємо помітну перевагу. Але при застосуванні розв'язання рівнянь чисельними методами потрібно особливу увагу приділити перевірці коректності

програмної реалізації. Для цього часто використовують порівняння результатів чисельного та «аналітичного» розв'язання підстановкою відповідних умов задачі, коли обидва метода є прийнятними. Крім того, необхідно перевіряти збіжність та стійкість розрахунків варіюванням відповідно деталізації розрахунків (наприклад кількості кроків) та умов задачі.

При розв'язанні задач, які призводять до необхідності застосування диференціальних рівнянь, зазвичай використовують додаткові умови:

- початкові – коли досліджується динаміка системи, тобто зміна її стану в часі;
- крайові – пошук розв'язку, який вдовольняє властивостям на певній геометрії задачі, наприклад отримання структури електромагнітних хвиль у хвилеводі.

Задача з початковими умовами є дуже популярною в моделюванні. Вона ще називається задачею Коші і розглядається математиками як одна з базових задач у теорії диференціальних рівнянь. У цій роботі ми розглянемо саме таку задачу, застосувавши її до рівняння руху певного об'єкту.

Для чисельного розв'язання диференціальних рівнянь в реальних задачах зазвичай використовують метод Рунге-Кутти, але в даній роботі обмежимось найпростішим його варіантом (з якого він і бере свій початок) – методом Ейлера, основним недоліком якого є ненайкраща збіжність. Але він максимально простий для розуміння при первинному ознайомленні, його легко реалізувати навіть у таблиці Excel або аналогу. Отже, спочатку розраховуємо похідні через підстановку у рівняння значень змінних з попереднього кроку, потім перераховуємо значення змінних для нового кроку додаючи їхній приріст, який обчислюється з щойно розрахованих похідних.

Завдання

1. Реалізувати чисельне моделювання поведінки простого фізичного об'єкта на основі розв'язання рівняння руху методом Ейлера.
2. Перевірити збіжність розв'язку.

Завдання цієї лабораторної не має окремих варіантів.

Хід виконання

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_12 у вказаній викладачем папці.

1.2. Реалізувати початкову структуру проєкту за зразком на рис.12.1.

```
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
using namespace std;

const double g = 9.81;          // м/(с*с)
double relief(double X)
{
    //return 0;
    return -0.01 * X + -2.5 * sin(X / 10) + 1.5 * sin(X / 13);
}
//---------------
double calcDistance(double alpha, double V, double Y0, double dt)
{
}

int main()
{
    // початкові умови
    double V = 10;            // швидкість каменя, м/с
    double alpha = 20;        // кут кидка у градусах
    double dt = 0.1;          // с
    double Y0 = 2;            // початкова висота, м

    printf("--------------------\n");
//...
    printf("--------------------\n");
    system("pause");
}
```

Рис.12.1

2. Розрахунок «дальності кидання каміння»

2.1. Реалізувати функцію calcDistance, яка повертає дальність кидання камінця за кутом кидання alpha, початковою висотою, з якої кинули камінець Y0, та початковою швидкістю V (рис.12.2). Опором повітря знехтувати.

2.2. Врахувати вплив рельєфу (його задає функцій relief).

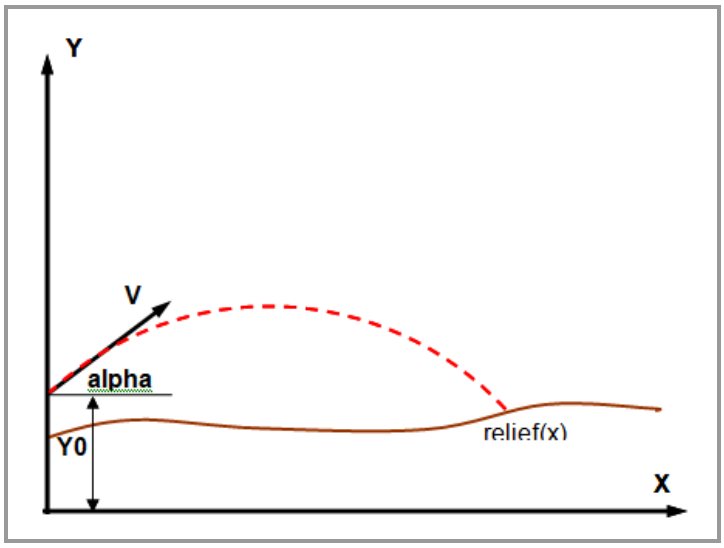


Рис.12.2

3. Залежність результату від кроку за часом dt та кута кидання

3.1. Вивести на консоль залежність дальності кидання від кроку за часом.

3.2. Для значення кроку за часом, який на вашу думку забезпечує достатню точність, вивести на консоль залежність дальності кидання від початкового кута.

3.3. Показати результати викладачу.

Цією лабораторною роботою ми завершили ознайомлення з використанням процедурного програмування, в тому числі і для не зовсім простих задач, які використовуються при математичному моделюванні. Але в сучасних умовах для того, щоб зробити проєктування задач високої складності більш зручним та ефективним, часто використовується дещо інша парадигма – об’єктно-орієнтоване програмування (ООП), якому буде присвячене останнє завдання.

Лабораторна робота №13. Розробка класу

Мета роботи: Навчитись будувати клас із заданою функціональністю.

Загальні відомості

У багатьох попередніх завданнях мали можливість потренуватись у використанні в одному проєкті декількох функцій. І нібито такого синтаксису повністю вистачало – усе, що було потрібно у функцію передавали через параметри, результати повертали через return або, якщо їх багато, також через параметри. При цьому окремі дані зберігали або в окремих змінних, або у масивах.

Але при розробці великих за обсягом проєктів, перспективами яких для реальних практичних задач постійно підлякували в попередніх роботах, при такій парадигмі програмування можуть виникати суттєві незручності. Певною проблемою, якщо проявити деяку прозорливість, в першу чергу є плутанина у можливій великій кількості імен змінних та функцій. До цього можна додати можливі надзвичайно довгі списки параметрів функцій, а до них треба забезпечувати ще й повну відповідність при написанні виклику. Для скорочення списків параметрів у функціях можна використати передачу даних через глобальні змінні, але велика їх кількість викликає навіть більшу плутанину.

Але рішення є. Допомогти може групування, групування і ще раз – групування. Програми не пишуть заради самих програм, вони повинні відповідати чомусь корисному, отже – реальному. А навколишній світ є об’єктним. І кожен об’єкт має певні власні характеристики та дії, що враховують ці характеристики. Ось вона – оболонка, яка охоплює щось всередині. А за однаковістю внутрішньої будови об’єкти можна згрупувати в окремі множини – класи.

В програмуванні, правда, порядок дій обернений – не збирають докупи щось з того, що вже є, а спочатку описують будову майбутнього представника певного класу, тобто створюють таку-собі «декларацію про наміри». В цю «декларацію» включається сукупність «елементарних» комірок для даних, тобто змінних відповідних типів (у класі їх називають полями). Таке групування даних в певній оболонці при описі об’єкта дозволяє кожен з них описувати не одним значенням, як у випадку звичної змінної, а багатьма і до того ж різними за

типами (на відміну від масиву). До цього додають функції, специфічні для обробки даних саме представника цього класу (функції в класі називають методами). Оскільки функція згрупована з даними в спільній оболонці, доступ в ній до полів отримує простоту використання глобальних змінних – їх не потрібно включати у рядок виклику.

Якщо не вдаватись до статичних членів класу (не будемо лізти в хащі, залишивши це для лекції), опис класу фактично створює новий складний тип (в доповнення до класичного набору базових типів від самої мови програмування). Відповідно, як можна оголосити декілька різних змінних або масив, наприклад, цілого типу, так само можна оголосити необхідну кількість екземплярів описаного класу – змінних нашого нового типу, або масив, комірки якого будуть представниками класу. Можна також використати клас, як тип поля в іншому класі – наприклад, масив «колес» є частиною «автомобіля».

А можна ще описувати певні класи, як подальше ускладнення раніше описаних. І якщо згадати біологію або хімію, то можна ще й побачити деревоподібну ієрархію такої надбудови. Це називається «ієрархія класів», також залишимо її для лекції. Зараз у нас практична частина курсу і ставимо акценти саме на можливостях використання.

При описі класу пам’ятаємо про його спеціальний метод – конструктор. Якщо функція (метод) може бути викликана багато разів, конструктор в даному екземплярі викликається один раз, до того ж автоматично, при створенні цього екземпляру. Призначенням конструктора є підготувати екземпляр до майбутнього застосування внесенням в його поля певних значень (як при ініціалізації звичайної змінної). Оскільки окремі екземпляри зазвичай відрізняються такими значеннями, найбільш корисним є конструктор з параметрами, через які задається така конкретика. Якщо конструктор в процесі «життя» екземпляру працює першим, то деструктор спрацьовує останнім. Його наповнення найчастіше використовується для «прибирання сміття» (в C++ немає автомата прибирання), тобто видалення блоків даних, якщо такі створювались екземпляром динамічно.

Мабуть, найпростіших відомостей про класи та їх екземпляри достатньо. Тепер про чергу, з якою буде потрібно працювати в цій роботі.

Черга (англ. queue) це лінійна структура даних, чим трохи нагадує масив. До речі, в деяких мовах її функціональність і реалізується масивом, а в C / C++ її можна, хоч і трохи незграбно, певним чином реалізувати на основі масиву. Нові елементи черги додаються з одного боку (у хвіст), а вибираються (одночасно з видаленням) з іншого (з голови). Тобто реалізується принцип FIFO (First In, First Out – «перший прийшов, перший вийшов») на кшталт черги до каси в універсамі. Черга є дуже важливою для програмної обробки даних, які надходять нерівномірно в часі, тобто колись їх стає більше, колись менше. І саме черга балансує цю нерівномірність.

Чергу можна реалізувати і без класів, але при цьому отримаємо певний хаос змінних, масивів та функції. Тому є слушним описати клас «черга». Більше того, елементи даних, що в неї вносяться, також можуть бути описані не простим типом, а також як деякий клас. Для забезпечення необмеженості кількості даних, що накопичується в черзі, краще її елементи утворювати динамічно при їхньому додаванні. Але для цього з корисними даними у кожному елементі черги треба пов’язувати посилання на іншій (наступний) – вже для цього треба використовувати групування в оболонку, тобто структуру або клас.

Спробуємо реалізувати такий клас. Хтось може заперечити – навіщо, оскільки черга реалізована в STL – стандартній бібліотеці C++, яка дуже ефективно доповнює стандартну бібліотеку C. Але слід пам’ятати про те, що інженер повинен не тільки користуватись готовими «цеглинами», а й розробляти нові, тобто прагнути розбиратись в їхній будові. Саме тому багато навчальних задач є відтворенням того, що є вже готовим і широковживаним.

Завдання

Розробити клас, який реалізує чергу, елементами якої є екземпляри іншого класу. Завдання цієї лабораторної не має окремих варіантів.

Хід виконання

1. Загальна підготовка проєкту

1.1. Запустити IDE Code::Blocks, створити новий проєкт «Console application», обравши мову C++, з назвою task_13 у вказаній викладачем папці.

1.2. Реалізувати початкову структуру проєкту за зразком на рис.13.1.

```
#include <iostream>
using namespace std;

class qe
{
public:
    static int guid;
    //....
    qe(double newdata)
    {
    }
};
int qe::guid=0;

class Q
{
public:
    qe* Front;
    qe* Back;
    qe* Get()
    {
    }
    void Set(qe* newel)
    {
    }
    Q()
    {
    }
};

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

Рис.13.1

1.3. Підготувати у функції main рядки, необхідні для перевірки працездатності проєкту за зразком на рис.13.2.

```
int main()
{
    qe* e1=new qe(10);
    qe* e2=new qe(20);
    qe* e3=new qe(30);
    qe* e4=new qe(100);
    qe* e5=new qe(200);

    qe* e=NULL;
    Q q;
    q.Set(e1);
    q.Set(e2);
    q.Set(e3);

    e= q.Get();
    e= q.Get();

    q.Set(e4);
    q.Set(e5);

    e= q.Get();
    e= q.Get();
    e= q.Get();
    e= q.Get();
    e=NULL;

    return 0;
}
```

Рис.13.2

2. Клас, який є елементом майбутньої черги

2.1. Описати поля цього класу (його заготовка вже є!). Кожен екземпляр цього класу повинен мати цілочисельний унікальний ідентифікатор (uid), деяке умовне «корисне значення», роль якого буде

відігравати дійсне число, та посилання на наступний елемент черги.

2.2. Реалізувати конструктор цього класу з параметром, який буде задавати «корисне значення» нового екземпляру.

2.3. Перевірити налагоджувачем працездатність, в тому числі присвоєння коректного значення uid.

3. Клас «черга», яка працює з елементами описаного у п.2. класу

Повним виконанням є динамічна черга. Спрощеним (з відповідним зниженням оцінки) є статична черга на основі масиву, який використовується для збереження даних.

STL використовувати заборонено!

3.1. Реалізувати метод, який додає у хвіст черги новий елемент з певним «корисним значенням».

3.2. Реалізувати метод, який виймає елемент з голови черги.

3.3. Реалізувати конструктор черги, який виконує попередню ініціалізацію її полів.

3.4. Перевірити налагоджувачем працездатність.

3.5. Показати викладачу.

Якщо не вдалось повністю зробити останнє завдання, нічого страшного немає. Вже добре те, що зробили спробу. Воно дійсно є найскладнішим у цьому практикумі – на те воно й останнє. Але якщо пізніше доведеться мати справу з текстом програми, реалізованої в парадигмі ОПП, як мінімум, щось згадаєте та зможете розібратись. А може після додаткового тренування вийде і використати на практиці такий, більш сучасний, спосіб побудови складних проєктів.

Література

1. Васильєв О. Програмування на С++ в прикладах і задачах . – Київ: Ліра-К, 2019. – 381,

2. Креневич А.П., Обвінцев О.В. С у задачах і прикладах. – Київ: Київський університет,2012.-212 с.

3. Грицюк Ю., Рак Т. Програмування мовою С++.– Львів: ЛДУ БЖД, 2011.- 146 с.

4. M. McGrath. Programming in Easy Step – In Easy Steps, 2022. – 192 p.

5. Herb Schildt's C++ Programming Cookbook. – McGraw Hill 2008.– 528 с.

6. M. Bancila. Modern C++ Programming Cookbook: Master Modern C++ with comprehensive solutions for C++23 and all previous standards.– Packt Publishing, 2024. – 816 .p.

7. М.І.Романюк, О.А.Батіна. Обчислювальна математика. – Київ: КПІ, 2019. – 193 с. (https://ela.kpi.ua/bitstream/123456789/27922/1/Obchysliuvalna-matematyka_konsp.lekts.pdf)