**Практична робота 10. Написання програм лінійної, розгалуженої та циклічної структури з вкладеними циклами, використанням математичних функцій та виконанням операцій введення- виведення**

**Мета**: навчитися писати програми лінійної, розгалуженої та циклічної структури з вкладеними циклами та використанням математичних функцій та виконанням операцій введення - виведення **на мові С++** в **консольному режимі.**

**Методичні вказівки щодо організації самостійної роботи студентів**

1. Запустити середовище програмування С++ .
2. Записати **програму**, що виконує 3 завдання з пп.3.1-3.3 (це ЛР№ 8+нове завдання). В першому рядку кожної програми записати

*// МНТ/ЕТ Група № Прізвище Номер ЛР*

вказавши номер своєї групи та своє прізвище.

Програма повинна запитати номер завдання (число 1, 2, 3 або 0 для закінчення) і в залежності від введеного значення виконувати відповідне завдання. Якщо введений 0 – програма припиняє роботу.

Вхідні дані ввести, а результати вивести, використовуючи потокове введення-виведення даних. Виводяться (змінні ***a*** або ***N*** перед початком розрахунку*, а* ***х, y*** та відповідні повідомлення на кожній ітерації/кроку циклу***).***

1. Завдання:
   1. Написати код, що виконує розрахунок ***y*** підсумку ***N*** перших членів з точністю ε=10-3:



Підсумок по((значити як Y, значення ***N*** ввести перед початком роботи.

Проаналізуйте вираз підсумку та складіть компактну формулу розрахунку в циклі.

* 1. Написати код, що виконує розрахунок ***y*** підсумку ***N*** перших членів з точністю ε=10-3



* 1. . Обчислити і вивести на екран у табличному вигляді значення функції *f(x)* на заданому інтервалі зміни значень аргументу *х* від *xпоч* до *xкін* з кроком *h.*

Коефіцієнти *a, b, c –* дійсні числа. Значення *a, b, c, xпоч*, *xкін*, *h* вводити з клавіатури. **Передбачити перевірку допустимості введених значень**.



Вхідні дані ввести, а результати вивести, використовуючи потокове введення-виведення даних.

1. Запустити програму на виконання. Перевірити всі варіанти виконання завдання. При відсутності помилок текст програми надіслати викладачу.
2. При необхідності проведення консультації з питань виконання лабораторних та практичних робіт відправте відповідне повідомлення із запитаннями на електронну пошту викладача.

Результати надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді cpp-файлу з іменем у форматі

**<Номер групи><Номер лабораторної><Прізвище англійською>**

Наприклад, 31-01Ivanov.cpp.

Іншим рішенням є надсилання поштою посилання на текст програми за URL адреси, яку надає C++Shell, вказавши в темі листа, номер групи прізвище студента та номер ПР.

В темі листа вказати, номер групи, прізвище студента та номер ПР як "ПР№3".

**Строк відсилки ЛР для МНТ/ЕТ-41 19.03.2024**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, В темі листа вказати, номер групи, прізвище студента та номер ПР як "ПР№5 В темі листа вказати, номер групи, прізвище студента, номер ПР та фразу "Запитання".

**Запитання**

1. Які оператори називають операторами повторення (циклу)?
2. Назвітьоператори повторення (циклу) С++.
3. Наведіть загальний синтаксис циклу for.
4. Яку змінну називають лічильником циклу?
5. Чи може цикл for мати кілька лічильників?
6. Який цикл називають циклом з передумовою? Поясніть.
7. Наведіть загальний синтаксис циклу while.
8. Який цикл називають циклом з постумовою? Поясніть.
9. Наведіть загальний синтаксис циклу do while.
10. Яка різниця між циклом, що керується лічильником і циклом, що керується контрольним значенням?
11. Яке призначення оператору break?
12. Яке призначення оператору goto?
13. Яке призначення оператору continue?

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

Розглянемо декілька прикладів на розрахунок підсумку членів деякого ряду значень.

***Приклад 1.***



Тут чисельник зменшується на 2, а знаменник збільшується на 2, тобто формула буде мати вигляд.

**((R – (2+J))/(R +J)**

Цей запис означає, що J змінюється від 0 до N з кроком 2, а сам вираз для розрахунку надається в дужках.

***Приклад 2.***



Тут до чисельника додається число **j,** яке змінюється від 1 до ***N*** з кроком 1***,*** а до знаменника – число 3**j**. Тобто формула бути мати вигляд:

**((x+j)/ (x+3j))**

**Переривання циклу: оператори break, сontinue, return.**

У деяких випадках буває необхідно перервати повторення циклу.

Один із способів розв’язання цієї задачі – використання оператору **break**. Він призначений для передчасного виходу з тіла циклу. Оператор **break** перериває виконання тіла будь-якого циклу (for, do чи while) та передає управління наступному за циклом оператору. При цьому умова виходу з циклу перевіряється в будь-якому його місті.

Ще один спосіб переривання циклу – використання оператору **goto**, що передає управління певному оператору за межами тіла циклу.

Для переривання циклів, розміщених у функціях, можна використати оператор **return.** На відміну від оператору **break,** оператор **return** перериває не лише виконання циклу, але й виконання тієї функції, у якій розміщений цикл.

Розглянуті способи переривали виконання циклу. Але є ще процедура **сontinue,** яка перериває лише виконання поточної ітерації, поточного виконання тіла циклу та передає управління на наступну ітерацію. Тобто виконання оператора continue призводить до пропускання операторів, що слідують за ним у тілі циклу.

**Приклад програми**

**Завдання:** Вивести на екран літери английского алфавіту.

**#include <iostream>**

**#include <conio.h>**

**using namespace std;**

**void main()**

**{**

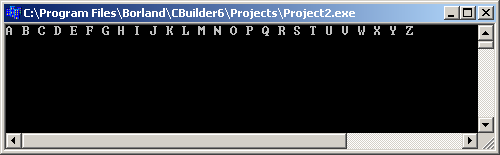
**char letter;** // лічильник циклу - змінна типу char

**for(letter=’A’;letter<=’Z’;letter++)** // цикл від A до Z з кроком 1

**cout << letter << " ";** // виведення літери

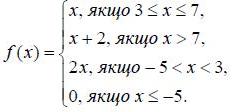
**getch();**

**}**

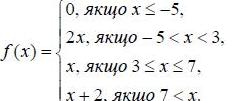
****

**Використання умовних операторів**

**Розглянемо приклад**. Написати фрагмент коду, що за дійсним *x* обчислює значення *f*(*x*) і присвоює його дійсній змінній *y*.



Перепишемо формулу обчислення *f*(*x*) у еквівалентному вигляді.



Якщо код писати безпосередньо за формулою, то такий фрагмент коду:

**if (x<=-5) y=0;**

**if (-5<x && x<3) y=2\*x;**

**if (3<=x && x<=7) y=x;**

**if (7<x) y=x+2;**

Цей код є правильним, але *неоптимальним* за кількістю виконуваних операцій. Якщо значенням змінної **x** є **-9.0**, то обчислюються всі чотири умови, хоча з погляду математики зрозуміло, що за істинності умови **x<=-5** решта умов хибні. Отже, модифікуємо фрагмент коду.

**if (x<=-5) y=0;**

**else if (-5<x && x<3) y=2\*x;**

**else if (3<=x && x<=7) y=x;**

**else if (7<x) y=x+2;**

Тепер за значення **-9.0** обчислюється тільки перша умова. Нехай значенням змінної **x** є **0.1**. Тоді вираз **x<=-5** є хибним, і обчислюється вираз **-5<x&&x<3**. Однак **x<=-5** хибний, тому **-5<x** є істинним! Отже, значенням виразу -**5<x&&x<3** є значення **x<3**. Міркуючи так само далі, отримуємо ще один варіант.

**if (x<=-5) y=0;**

**else if (x<3) y=2\*x; // тут значення -5<x істинне**

**else if (x<=7) y=x; // тут значення 3<=x істинне**

**else y=x+2; // тут значення 7<x істинне**

Зауважимо: наступний фрагмент коду для нашої задачі є *помилковим*.

**if (x<=-5) y=0;**

**if (x<3) y=2\*x;**

**if (x<=7) y=x;**

**else y=x+2;**

Якщо значенням **x** є **1.0**, то умова **x<3** істинна, тому спочатку виконується присвоювання **y=2\*x**. Проте потім перевіряється умова **x<=7**, виявляється істинною, і виконується **y=x**, що, вочевидь, є помилковим.

Дуже часто інструкції розгалуження є частиною інших розгалужень, тому їх записують "східцями", зсуваючи вкладену інструкцію праворуч, наприклад, таким чином:

**if (*умова1*)**

**if (*умова2*)**

***інструкція1***

**else**

***інструкція2***

**else ...**

Інколи виникають довгі ланцюги розгалужень, в яких за словами **else** йдуть наступні розгалуження з **if** на початку. Краще записувати їх у такому вигляді:

**if (*умова*)**

***інструкція***

**else if (*умова*)**

***інструкція***

**else if (*умова*)**

**...**

**ДЛЯ ДОВІДКИ**

**Математичні функції (**довідково**)**

| **Прототип функції** | **Ім’я** | **Призначення** |
| --- | --- | --- |
| Double sin (double \_х); | **sin (x)** | синус x (в радіанах) — **sin x** |
| Double cos (double \_x); | **cos (x)** | косинус x (в радіанах) — **cos х** |
| Double tan (double \_x); | **tan (x)** | тангенс х (в радіанах) — **tg х** |
| Double asin (double \_x); | **asin (x)** | арксинус х — **arcsin х** |
| Double acos (double \_x); | **acos (x)** | арккосинус х — **arcos х** |
| Double atan (double \_x); | **atan (x)** | арктангенс х — **arctg х** |
| Double atan2 (double \_y, Double\_x); | **atan2 (y,x)** | арктангенс у/х — **arctg (у/х)** |
| Double sinh (double \_x); | **sinh (x)** | синус гіперболічний х — **sh х** |
| Double cosh (double \_x); | **cosh (x)** | косинус гіперболічний х — **ch х** |
| Double tanh (double \_x); | **tanh (x)** | тангенс гіперболічний х — **th х** |
| Double log (double \_x); | **log (x)** | натуральний логарифм х — **ln х** |
| Double log10 (double \_x); | **log10 (x)** | десятковий логарифм х — **log х** |
| Double exp (double \_x); | **exp (x)** | піднесення е до степеня х — **ех** |
| Double pow (double \_x, double\_y); | **pow (x,y)** | піднесення х до степеня у — **ху** |
| Double pow 10 (int \_p) | **pow10 (p)** | повертає **10р** |
| Double sqrt (double \_х); | **sqrt (x)** | корінь iз x, x > 0 |
| Double hypot (double\_x, double\_y); | **hypot (x,y)** | корінь із (х2+у2) |
| Double fabs (double \_\_x); | **fabs (x)** | абсолютне значення х — |х| типу **double** |
| int abs (int \_x); | **abs (x)** | абсолютне значення х — |х| типу **int** |
| long labs (long \_x); | **labs (x)** | абсолютне значення х — |х| типу **long** |
| Double fmod (double \_\_x, double\_y); | **fmod (x,y)** | залишок від ділення х на у |
| Double ceil (double \_\_x); | **ceil (x)** | округлення до більшого |
| Double floor (double \_x); | **floor (x)** | повертає найближче ціле, не більше за х |
| Double modf (double \_x, double); | **modf(x,&p)** | виділяє цілу й дробову частинні числа |
| Double atof(const char\* \_s); | **atof (s)** | перетворює рядок символів у число з плаваючою крапкою |

У бібліотеці **cmath** означено константи з іменами **M\_PI** (число π), **M\_PI\_2** (π /2), **M\_PI\_4** (π /4), **M\_1\_PI** (1/ π), **M\_E** (число *e*), **M\_LN2** (ln 2), **M\_LN10** (ln 10) і деякі інші. Щоб користуватися ними, необхідно перед підключенням бібліотеки **cmath** записати директиву

**#define \_USE\_MATH\_DEFINES** (define – означити).

**Приклад**. Надана програма виводить значення математичних констант π та *e*.

**#include <iostream>**

**#define \_USE\_MATH\_DEFINES**

**#include <cmath>**

**using namespace std;**

**int main() {**

**cout<<"pi="<<M\_PI<<endl;**

**cout<<"e="<<M\_E<<endl;**

**cout<<endl;**

**}**

**ЗАСТОРОЖЕННЯ ЩОДО ВИКОРИСТАННЯ**

**ЦИКЛІВ ТА ОПЕРАТОРУ GOTO**

## Оператор goto

**Оператор goto** — це **оператор управління потоком виконання програм**, який змушує центральний процесор виконати перехід з однієї ділянки коду в іншу (тобто здійснити стрибок). Інша ділянка коду ідентифікується за допомогою **мітки**. Наприклад:

**#include <iostream>**

**#include <cmath> // для функції sqrt()**

**using namespace std; *// простір імен***

**int main()**

**{**

**double z;**

**tryAgain: // це мітка**

**cout << "Enter a non-negative number: ";**

**cin >> z;**

**if (z < 0.0)**

**goto tryAgain; // а це оператор goto**

**cout << "The sqrt of " << z << " is " << sqrt(z) << endl;**

**return 0;**

**}**

У цій програмі користувачеві пропонується ввести додатне число. Однак, якщо користувач введе від’ємне число, то програма, використовуючи оператор goto, виконає перехід назад до мітки tryAgain. Потім користувачеві знову потрібно буде ввести число. Таким чином, ми можемо постійно просити користувача ввести число, поки він не зробить це коректно.

Раніше ми розглядали два типи області видимості: **локальна** (або ще “блокова”) і **глобальна** (або ще “файлова”). Мітки використовують третій тип області видимості: **область видимості функції**. Оператор goto і відповідна мітка повинні знаходитися в одній і тій же функції.

Існують деякі обмеження на використання операторів goto. Наприклад, ви не зможете перестрибнути вперед через змінну, яка ініціалізована в тому ж блоці, що і goto:

**int main()**

**{**

**goto skip; // стрибок вперед заборонений**

**int z = 7;**

**skip: // мітка**

**z += 4; // яке значення буде в цій змінній?**

**return 0;**

**}**

Загалом програмісти уникають використання оператора goto в C++ (і в більшості інших високорівневих мов програмування). Основна проблема з ним полягає в тому, що він дозволяє програмісту керувати виконанням коду так, що точка виконання може довільно переміщуватися в коді. А це, в свою чергу, створює те, що досвідчені програмісти називають «спагетті-кодом». **Спагетті-код** — це код, порядок виконання якого нагадує тарілку зі спагеті (все заплутано і закручено), що вкрай ускладнює слідування та розуміння логіки виконання такого коду.

Як казав один відомий фахівець в інформатиці та в програмуванні, Едсгер Дейкстра: «Якість програмістів — це функція щільності використання операторів goto в програмах, які вони пишуть».

Оператор goto часто використовується в деяких старих мовах програмування, таких як Basic чи Fortran, або навіть в C. Однак в C++ goto майже ніколи не використовується, оскільки будь-який код, написаний з ним, можна більш ефективно переписати з використанням інших об’єктів в C++, таких як цикли, обробники винятків або деструктори (це все ми розглянемо у відповідних уроках).

**Правило: Уникайте використання операторів goto, якщо на це немає вагомої причини.**

## Лічильник циклу while

Часто потрібно щоб цикл виконувався певну кількість разів. Для цього зазвичай використовується змінна у вигляді лічильника циклу. **Лічильник циклу** — це цілочисельна змінна, яка оголошується з єдиною метою: рахувати, скільки разів виконався цикл.

Лічильникам циклу часто дають прості імена, такі як i, j чи k. Проте в цих іменах є одна серйозна проблема. Якщо ви захочете дізнатися, де у вашій програмі використовується лічильник циклу і скористаєтеся функцією пошуку i, j чи k, то в результаті отримаєте половину своєї програми, так як i, j та k використовуються в багатьох іменах. Отже, краще використовувати iii, jjj чи kkk в якості імен для лічильників. Вони більш унікальні, їх значно простіше знайти, і вони виділяються в коді. А ще краще використовувати «реальні» імена для змінних, наприклад, count або будь-яке інше ім’я, яке надає контекст використання цієї змінної.

Також для лічильників циклу краще використовувати тип **signed int**. Використання unsigned int може призвести до несподіваних результатів, наприклад:

**#include <iostream>**

**using namespace std; *// простір імен***

**int main()**

**{**

**unsigned int count = 10;**

**// Рахуємо від 10 до 0**

**while (count >= 0)**

**{**

**if (count == 0) cout << "blastoff!";**

**else cout << count << " ";**

**--count;**

**}**

**return 0;**

**}**

В програмі вище є помилка, вона являє собою нескінченний цикл. Вона починається з виведення 10 9 8 7 6 5 4 3 2 1 blastoff! як і передбачалося, але потім все йде шкереберть і починається відлік з 4294967295. Тому що умова циклу count >= 0 ніколи не буде хибною! Коли count = 0, то і умова 0 >= 0 має значення true, виводиться blastoff, а потім виконується декремент змінної count, відбувається **переповнення** і значенням змінної стає 4294967295. **Переповнення** (англ. “**Overflow**“) трапляється при втраті біт через те, що змінній не було виділено достатньо пам’яті для їх зберігання. І так як умова 4294967295 >= 0 є істинною, то програма продовжує своє виконання. А оскільки лічильник циклу є типу unsigned, то він ніколи не зможе бути від’ємним і цикл ніколи не завершиться.

**Правило: Завжди використовуйте тип signed int для лічильників циклу.**