**Лабораторна робота 02.** **Використання математичних функцій, введення даних для розрахунку та виведення результатів на консоль.**

**Мета**: навчитися використовувати математичні функції, операції введення – виведення даних в середовищі програмування **С++** в **консольному режимі.**

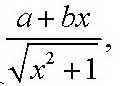
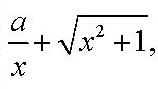
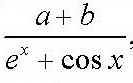
*Контрольні запитання*

1. Для чого слугують заголовні файли?
2. Як компілятор шукає розташування заголовних файлів?
3. Які маніпулятори використовуються для завдання довжини поля, дробової частини числа?
4. Який заголовний файл використовується для маніпуляторів завдання довжини поля, дробової частини числа?

**Методичні вказівки щодо організації самостійної роботи студентів**

1. Запустити середовище програмування С++.
2. Відкрити збережену на попередній ЛР у власній папці заготовку програми. До неї ввести програму, написану на попередній ПР, а саме:

Написати на мові С++ програму введення даних для розрахунку формул, наданих нижче, а також виведення результатів. Використати потокове введення та виведення даних. Значення a, b задати константами. Значення a, b задати константами.

1. До формул додати ще такі формули:

 ,, 

,, 

1. Результати розрахунку по кожній формулі зберігаються в окремій змінній.
2. Вивести результати таким чином:

* Перший рядок заголовок"Results".
* Другий рядок через табуляцію виводяться номери формул (1, 2 і т.д.).
* Третій рядок - через табуляцію результати розрахунку з використанням маніпуляторів **setw(w)** та **setprecision(d**), обравши довжину поля 7, кількість позицій у дробовій частині дійсних чисел – 2.

1. Запустити програму на виконання. Результати виконання надіслати викладачу.
2. Зберегти програму (наприклад, у текстовому файлі на своєму комп’ютері) для використання в подальших ЛР.

Першим рядком програми повинен бути коментар з прізвищем студента на назвою роботи (наприклад, // Батс ПР 4).

Результати у вигляді текстового файлу надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)

Файл повинен мати назву в такому форматі:

**АР<Номер групи><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**. Наприклад, **АРPTBD-2104P**buts.doc.

**Тему в заголовку листа записати**

**АР <Номер групи>-><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**

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.

**Строк виконання цієї роботи і тесту - 21.02.2021**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**АР<Номер групи>-Запитання-<Прізвище англійською>**.

**Теоретична частина**

##### ***Приведення типів в операціях +, –, \****

У виразах, в яких фігурують операції **+**, –, **\*** діють такі правила приведення типу результату:

* якщо обидва операнди мають цілий тип, то результат також буде цілого типу;
* якщо хоча б один з операндів має дійсний (з плаваючою комою) тип а інший цілий тип, то результат також буде дійсного типу;
* якщо один з операндів має тип float, а інший тип double, то результат буде типу double. Це зв‘язано з тим, що тип double потребує більше пам‘яті ніж тип float. У цьому випадку відбувається розширення типу float до типу double.

##### ***Пріоритет та асоціативність арифметичних операцій***

Арифметичні операції мають пріоритет та асоціативність такі, як зображено в таблиці.

|  |  |  |
| --- | --- | --- |
| **Знаки операцій** | **Найменування** | **Асоціативність** |
| **\* / %** | Бінарні, мультиплікативні | Зліва направо |
| **+ –** | Бінарні, адитивні | Зліва направо |

##### відмінність між бінарними та унарними операціями додавання (+) та віднімання (–)

Операції додавання (**+**) та віднімання (–) можуть бути як бінарними, так і унарними.

Бінарні операції **+** та **–** використовуються у виразах при проведенні обчислень.

Унарні операції **+** та **–** використовуються для позначення знаку числа (додатне число або від‘ємне число).

**Приклад.**

int a, b;

a = -8; // унарна операція '-', позначає знак числа

b = +9; // унарна операція '+', b = 9

a = b - 5; // бінарна операція '-', використовується у виразі для обчислення

##### особливості використання операції % (остача від ділення

Операція **%** використовується над цілими операндами. Операція **%** дозволяє отримати остачу від ділення цілих операндів.

**Приклад.**

// Операція % - взяття остачі від ділення

int a, b;

int c;

a = 3;

b = 5;

c = a % b; // c = 3

a = 8;

b = 4;

c = a % b; // c = 0

c = 12 % 35; // c = 12

c = 35 % 12; // c = 11

c = -5 % -3; // c = -2

##### ***Особливості використання операції / (ділення)***

Операція ділення має свої особливості, які полягають в наступному:

* якщо два операнди мають цілочисельний тип, то результат повертається цілого типу. У цьому випадку відбувається ділення націло. Остача від ділення відкидається;
* якщо один з операндів має тип з плаваючою комою, тоді результат має також тип з плаваючою комою.

**Приклад.**

// Операція ділення

int a, b;

int c;

float x;

a = 8;

b = 3;

c = a / b; // c = 2

x = a / b; // x = 2.0

x = a / (float)b; // x = 2.666667

x = 17.0 / 3; // x = 5.666667

x = 17 / 3; // x = 5.0

##### ***Особливості використання операторів інкременту та декременту в програмах на C++***

У мові C++ визначено два оператора, що здійснюють збільшення або зменшення цілочисельної величини на 1:

* оператор **++** – інкремент;
* оператор **––**  – декремент.

Ці оператори є унарними. Вони вимагають одного операнда. Ці оператори можуть розміщуватись до та після операнда.

**Оператор інкременту ++** збільшує значення операнду на 1. Наприклад, рядок

x = x + 1;

є аналогічний рядку

x++;

або

++x;

Так само, **оператор декременту —** зменшує значення операнду на 1. Наприклад, рядок

x = x – 1;

можна записати

x--;

або

--x;

##### ***Приклади застосування операторів інкременту (++) та декременту (––)***

Фрагмент коду, що пояснює роботу операторів **++** та **––**.

// оператори інкременту (++) та декременту (--)

int a, b;

a = 10;

b = a++; // b = 10; a = 11

a = 10;

b = ++a; // b = 11; a = 11

a = 10;

b = a--; // b = 10; a = 9

a = 10;

b = --a; // b = 9; a = 9

##### ***Відмінність між виразом ++x (***[***––***](https://www.bestprog.net/uk/2017/10/13/operators-of-increment-and-decrement-the-complex-assignment-operators_ua/#q02)***x) та виразом x++ (x***[***––***](https://www.bestprog.net/uk/2017/10/13/operators-of-increment-and-decrement-the-complex-assignment-operators_ua/#q02)***)***

Відмінність між префіксною та постфіксною формами операторів інкременту (**++**) та декременту ([**––**](https://www.bestprog.net/uk/2017/10/13/operators-of-increment-and-decrement-the-complex-assignment-operators_ua/#q02)) проявляється, коли ці оператори беруть участь в операції присвоювання.

Якщо вираз **++x** використовується в операторі присвоєння

y = ++x;

то він працює у такому порядку:

* спочатку значення x збільшується на 1, а потім результуюче значення присвоюється змінній y.

Якщо виконати вираз

y = x++;

то він працює у такому порядку:

* спочатку змінній y присвоюється значення x, а потім значення x збільшується на 1.

##### ***Складені оператори присвоювання, що використовуються в C++***

У мові C++ можна використовувати складені оператори присвоювання. Ці оператори є зручними, коли в програмі використовуються довгі імена змінних. У цьому випадку відпадає необхідність зайвий раз вводити довге ім’я змінної.

**Загальний вигляд** складеного оператора присвоювання наступний:

ім‘я\_змінної ***operation***= вираз;

де

* *ім‘я\_змінної* – безпосередньо ім‘я змінної, якій присвоюється значення;
* ***operation*** – одна з операцій **+**, **–**, **\***, **/**, **%**, **&**, **|**, **^**, **<<**, **>>**.

Мова C++ підтримує такі складені оператори присвоювання:

**+=**, **-=**, **\*=**, **/=**, **%=**, **&=**, **|=**, **^=**, **<<=**, **>>=**

##### Приклади використання складених операторів присвоювання

// складені оператори присвоювання

float x, y;

int a, b;

// +=, -=

a = 8;

b = 5;

a += b; // a = a + b = 13

b -= 4; // b = b - 4 = 1

// \*=, /=

x = 4;

y = 5;

x \*= y; // x = x \* y = 4 \* 5 = 20

y /= 2.5; // y = y / 2.5 = 2.0

// &=, |=

a = 8;

b = 3;

a &= b + 5; // a = a & b + 5 = a & (b+5) = 8

a |= b; // a = a | b = 11

// >>=, <<=

a = 34;

a >>= 1; // a = a >> 1 = 17

a = 6;

a <<= 3; // a = a << 3 = 48

// %=

b = 15;

b %= 6; // b = b % 6 = 3

##### ***Застосовання операцій інкременту та декременту до типів з плаваючою комою (float, double, long double)***

Зі змінними типів з плаваючою комою операції інкременту та декременту працюють так само, як і зі змінними цілого типу.

**Приклад.**

float x;

x = 24.5;

x--; // x = 23.5

##### застосовувати операції інкременту та декременту до символьного типу (char)

Оскільки, символьний тип char неявно відноситься до цілочисельних типів, то до змінних символьного типу можна застосовувати операції інкременту та декременту.

**Приклад.**

char c;

c = 'x';

c++; // c = 'y'

c = '6';

--c; // c = '5'

##### ***Призначення у програмі на C++ операції sizeof***

Операція sizeof призначена для визначення розміру типу даних, змінної базового типу, змінної структурного типу, числового значення, рядкового значення тощо.

Загальний вигляд операції sizeof:

sizeof(*тип\_або\_змінна*)

де

* *тип\_або\_змінна* – назва типу даних або змінної (об‘єкту), що використовується на даний момент в програмі.

##### ***Приклади використання операції sizeof для базових типів та числових значень***

У наведеному нижче фрагменті коду наведено приклад визначення розміру змінної базового типу, числового значення, рядкового значення або результату виразу.

// операція sizeof

int a;

short int b;

float x;

double y;

long double z;

int size;

size = sizeof(b); // size = 2

size = sizeof(int); // size = 4

size = sizeof(a); // size = 4

size = sizeof(x); // size = 4

size = sizeof(double); // size = 8

size = sizeof(z); // size = 8

size = sizeof(long double);

size = sizeof(bool); // size = 1

size = sizeof(true); // size = 1

size = sizeof(28); // size = 4 - як тип int

size = sizeof(9.8 + 5); // size = 8 - як тип double

size = sizeof("Hello world!"); // size = 13

size = sizeof('\n'); // size = 1

##### ***Тернарна операція ? :***

Тернарна операція **? :** може замінювати оператор [**умовного переходу if … else**](https://www.bestprog.net/uk/2017/08/02/conditional-jump-operator-if-2_ua/). Загальний вигляд операції ? : наступний:

***вираз1* ? *вираз2* : *вираз3***

де

* *вираз1* – будь-який логічний вираз, результатом якого є значення true або false;
* *вираз2* – вираз, що буде обчислений, якщо значення *вираз1* = true;
* *вираз3* – вираз, що буде обчислений, якщо значення *вираз1* = false.

##### Приклади використання операції ? :

У нижченаведеному фрагменті коду обчислюється мінімальне значення між двома змінними a та b:

// операція ? :

// мінімальне значення між двома числами

int a, b;

int min;

a = 15;

b = 8;

min = a > b ? b : a; // min = 8