**Тема 3-4. Арифметичні операції. Поняття вхідного та вихідного потоку, найпростіші математичні функції.**

**Арифметичні операції**

**Операції, операнди, вирази**

Обчислювальні дії в програмуванні називаються **операціями**. Операції застосовуються до **операндів**, тобто значень. Застосування операцій до значень описують у вигляді **виразу** (*expression*). Послідовність застосування операцій називається **обчисленням** виразу й має результатом **значення виразу**.

Операції у виразі позначаються знаками (**операторами**), а значення – константами та іменами змінних. У виразі також можуть бути дужки, що визначають порядок застосування операцій. Найпростішими виразами є ті, що не містять операцій, тобто константи та імена змінних.

**Приклад 1**.

* Вираз **2+2** означає: додаються 2 та 2 і значенням виразу є 4;
* вираз **2\*radius** – 2 множиться на значення змінної **radius** і значенням виразу є подвоєне значення цієї змінної;
* **1+2\*3** – множаться 2 та 3, отриманий добуток 6 додається до 1 ізначенням є 7;
* **(1+2)\*3** – додаються 1 і 2, їх сума 3 множиться на 3 та значенням є 9.

Два останні вирази демонструють, як дужки впливають на порядок операцій. Вираз має подвійну семантику – послідовність операцій з операндами, а також значення, що є результатом цієї послідовності.

Операція з одним або кількома значеннями, результатом якої є число, називається **арифметичною**. Спочатку розглянемо тільки деякі з багатьох арифметичних операцій мови С++.

Операції **додавання**, **віднімання**, **множення** й **ділення** мають знаки відповідно **+, -, \*, /**. Результатом операції з *цілими* числами є *ціле* число, з *дійсними* – *дійсне*. Наприклад, значенням виразу **4/2** є ціле число **2**, а виразу **4.0/2.0** – дійсне **2.0**.

Знак **–** позначає як двомісну операцію віднімання, так і одномісну операцію "мінус": **-32768**, **-(2+3)**. Знак **+** також може позначати одномісну операцію.

Результатом ділення **/** цілих чисел є *ціла частка* від ділення з остачею, наприклад, вираз **7/3** має значення **2**. *Цілу остачу* від ділення обчислює операція **%**: значенням виразу **7%3** є **1**. Зауважимо: знак остачі збігається зі знаком діленого, наприклад, обидва вирази **-7%3** та **-7%3** мають значення **-1**, а вираз **-7%-3** – значенняº **1**.

Результатом ділення дійсних чисел є число в його дійсному зображенні, наприклад, значенням виразу **7.0/3.0** є деяке наближення до числа **2.33…**, а виразу **6.0/3.0** – число **2.0**. Операція **%** до дійсних чисел незастосовна.

Виконання операції **/** або **%** з дільником 0 призводить до аварійного завершення програми.

Одномісна операція обчислює цілу *кількість байтів*, зайнятих її операндом (дані типу **char** займають 1 байт, типу **int** – 4 байти, **double** – 8). Отже, під час виконання інструкції

**cout<< sizeof 'A' << ' ' << sizeof 1 << ' ' << sizeof 0.0 << endl;**

отримаємо **1 4 8**.

**Старшинство операторів і порядок виконання операцій**

Мова С++ в основному відповідає угодам математики про порядок застосування операцій у виразах. Це дозволяє не записувати зайві дужки, наприклад, **1-2\*3** означає те саме, що й **1-(2\*3)**. На порядок обчислення виразу за відсутності дужок впливає **старшинство** (*precedence*), або **пріоритет**, операторів: якщо поруч із позначенням операнда записано два оператори, то спочатку виконується операція, що відповідає старшому оператору (з вищим пріоритетом). Наприклад, пріоритети **\*** та **/** однакові й вищі за **+** і **-**. Одномісні оператори старші за двомісні, а двомісні **\***, **/**, **%**, -, **+**, старші за всі інші двомісні, у тому числі присвоювання.

Окрім пріоритетів, оператори мають властивості право- або лівобічного зв'язування. У мові С++ усі двомісні оператори, окрім присвоювань, мають властивість **лівобічного зв'язування**: якщо ліворуч і праворуч від позначення операнда записано знаки операцій з однаковим старшинством, то операнд зв'язується з оператором, указаним ліворуч (ця операція застосовується спочатку).

**Приклади 2.**

1. Значенням виразу **4+7/5** є **5**, оскільки спочатку обчислюється **7/5** із результатом **1**, а потім **4+1** із результатом **5**. Значенням виразу **(4+7)/5** є **2**, оскільки спочатку обчислюється операція в дужках **(4+7)** – її значення 11, а потім **11/5** із результатом **2**.

2. У виразі **sizeof 2.0+4** обчислюється **sizeof** із результатом **8**, потім додається **4**.

3. Значення виразу **4-3-2** дорівнює **-1**, оскільки спочатку обчислюється **4-3**, тобто **1**, а потім **1-2**; у виразі **2\*7%8** спочатку обчислюється **2\*7** (це **14**), потім **14%8**, тобто **6**.

4. Нехай дійсні змінні **a**, **b**, **c** зображують коефіцієнти квадратного рівняння *ax*2+*bx*+*c*=º0. Дискримінант рівняння визначається виразом **b\*b–4\*a\*c**. Присвоїмо його дійсній змінній **d**: **d=b\*b–4\*a\*c**.

Пріоритети операторів дозволяють не записувати зайві дужки, але зловживати цим не слід. Інколи необов'язкова пара дужок значно підвищує зрозумілість запису. Наприклад, у виразі **sizeof 2.0+4** пробіл між **sizeof** та **2.0** провокує людину спочатку (помилково) обчислити **2.0+4**, а потім **sizeof**. Проте **sizeof(2.0)+4** є очевидним.

**Збільшення та зменшення**

У циклічних обчисленнях дуже часто використовуються присвоювання вигляду **x=x+1** та **x=x-1**. Їх можна задати в скороченій формі за допомогою одномісних операторів **збільшення** (інкременту) **++** і **зменшення** (декременту) **--**. Ці оператори (і відповідні операції) мають **префіксну (++x, --x**) і **постфіксну** (**x++, x--**) **форми**.

Вираз із постфіксним оператором **x++** або **x--** змінює значення змінної **x** на **1**, але значенням самого виразу є значення **x** *перед зміною*. Вираз із префіксним оператором **++x** або **--x** теж змінює **x** на **1**, але значенням виразу є значення **x**, отримане *після зміни*. Ці відмінності виявляються, коли оператори **++** та **--** застосовуються всередині виразів.

Операції **++** та **--** виконуються швидше ніж відповідні присвоювання вигляду **x=x+1** та **x=x-1**, тому рекомендується використовувати саме їх. Операції **++** та **--** застосовні до змінних будь-якого з базових типів, хоча найчастіше їх використовують із цілими змінними.

Скрізь, де немає необхідності використовувати старе значення змінної, рекомендується з виразів вигляду **n++** та **++n** вибирати **++n**, оскільки він виконується швидше й простіше.

Спосіб і порядок обчислення виразу залежить від компілятора, тому краще записувати операції збільшення або зменшення в окремих виразах або інструкціях, а не у складі інших виразів. Наприклад, значення виразів **(n++)\*(n++)** та **(++n)\*(++n)** у різних системах програмування навіть можуть відрізнятися. Гарантовано лише те, що до значення змінної **n** двічі додається 1.

##### ***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

Директиви підключення ресурсів. Служать для підключення до тексту програми текстів *заголовних файлів* (мають стандартне розширення .h). ***Заголовний файл*** – це файл, що містить описи деяких функцій та оголошення типів даних і констант, які використовуються цими функціями. Список заголовних файлів визначається стандартом мови.

Існують два способи підключення директиви #include:

1. # include <ім'я> - підключення стандартних заголовних файлів;
2. # include "ім'я" - підключення заголовних файлів користувача.

Різниця між ними полягає в тому, де препроцесор буде шукати файли-ресурси, які необхідно підключити для повної підготовки коду програми до компіляції: якщо ім’я файла взяте в кутові дужки <…>, то пошук буде вестись в наперед визначених каталогах; якщо ім’я взяте в лапки, то препроцесор шукає його в поточному каталозі, де міститься і основний файл компілювання.

Слід зазначити, що багато стандартних заголовних файлів має порожнє розширення, для решти - традиційно використовують розширення h. Наприклад,

#include <iostream> *// підключення засобів потокового введення-виведення*

#include < stdio.h> *// підключення засобів форматованого введення-виведення*

Основні заголовні файли, які входять до складу стандартної бібліотеки, подані в табл.1.

Таблиця 1. Основні заголовні файли стандартної бібліотеки С/С++

| *Заголовний файл* | *Вміст* |
| --- | --- |
| <iosream.h> | прототипи для функцій стандартних вхідних і вихідних потоків |
| <iomanip.h> | прототипи функцій для маніпуляторів, що дозволяють форматувати потоки даних |
| <сmath.h> | прототипи функцій математичної бібліотеки |
| <stddef.h> | загальні визначення типів, що використовуються в С/С++ для виконання деяких обчислень |
| <stdio.h> | прототипи для функцій вводу/виводу стандартної бібліотеки і інформація, що ними використовується |
| <stdlib.h> | прототипи функцій для перетворення чисел в текст і тексту в числа, прототипи функцій розміщення в пам’яті, генерації випадкових чисел і інших сервісних функцій |
| <time.h> | прототипи функцій і типи для функцій управління часом і датою |

Розглянемо найпростіший приклад програми на C/*С++* :

**#include <iostream> *// директива препроцесора***

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

**int main() *// головна функція***

**{ cout << "Hello,world\n"; *// виведення повідомлення на екран***

**system("pause"); *// пауза***

**return 0; *// повернення результату (вихід із функції)***

**}**

Рядок #include (включити) означає, що препроцесор перед компіляцією програми має включити в неї вміст заголовного файлу iostream.h. У цьому файлі оголошено засоби потокового введення та виведення. Без включення цього файлу ім’я cout буде невизначеним і компілятор повідомить про цю помилку.

У другому рядку розташовано інструкцію компілятору «використати простір імен std». Не пояснюючи значення слів «простір імен», скажемо лише, що простір імен std є стандартним. У сучасних системах програмування мовою С++ у ньому описано всі бібліотечні засоби «останнього покоління». Проте не всі компілятори розуміють цю інструкцію, тому її можна писати не завжди.

Частина програми

**int main()**

**{ ...**

**}**

визначає головну функцію main.

Операція << ("помістити в") пише свій перший аргумент у другий (рядок "Hello,world\n" у стандартний потік виведення cout). У даному рядку символ “\n“ є управляючим символом переходу на новий рядок.

За інструкцією system("pause"); виконання програми призупиняється і на екрані з’являється повідомлення, що треба натиснути будь-яку клавішу. Після її натискання програма завершується. Завдяки використанню управляючого символу “\n“, повідомлення виводиться в новому рядку. Якби його не було, повідомлення з’являлося б відразу після слів Hello, world!. Цю інструкцію виконуть в компіляторах, які в консольному режимі закінчують работу і закривають консоль.

**Бібліотечні математичні функції та константи**

Деякі операції з числами позначають **викликами функцій**, тобто у вигляді ***f*(…)**, де ***f*** позначає певне ім'я. Розглянемо дві функції, означені в усіх реалізаціях мови С++. Для використання цих функцій у програмі необхідно підключити модуль **cmath**:

**#include <cmath>**

Одномісна функція **sqrt** обчислює квадратний корінь свого невід'ємного **дійсного** операнда. Значенням виразу **sqrt(2.0)** є приблизно **1.41421**, а **sqrt(4.0)** –значення **2.0**. До цілих чисел функція *незастосовна*.

Двомісна функція **pow** обчислює дійсний ступінь, основою якого є перший операнд, показником – другий. Наприклад, значенням виразу **pow(2.0, 3)** є **8.0**, виразу **pow(2, 0.5)** – приблизно **1.41421**. Результатом функції **pow** завжди є дійсне значення.

Функція **log** обчислює натуральний логарифм свого додатного дійсного аргументу, функція **log10** – десятковий логарифм.

Застосування функцій до цілих аргументів є помилковим.

Функція **fabs** обчислює дійсне значення |*x*| за дійсним аргументом *x*. Функція **abs** із бібліотеки **cstdlib** обчислює ціле значення |*x*| за цілим аргументом *x*; якщо аргумент дійсний; обчислене значення може відрізнятися від математичного.

**Приклади 3.**

1. Корінь із невід'ємного дискримінанта квадратного рівняння з дійсними коефіцієнтами **a, b, c** можна обчислити виразом **sqrt(b\*b–4\*a\*c)**, а дійсні корені рівняння – виразами**(-b+sqrt(b\*b–4\*a\*c))/(2\*a)** та **(-b+sqrt(b\*b–4\*a\*c))/(2\*a)**.

Дужки в знаменнику обов'язкові. Якщо їх не записати, то відбудеться не ділення, а множення на **a**.

2. Вираз **pow(b\*b–4\*a\*c,0.5)** позначає обчислення квадратного кореня з **b\*b–4\*a\*c**, вираз **pow(b,1.0/3.0)** – обчислення кубічного кореня з **b**, а обидва вирази **pow(2.0, 5)** та **pow(2, 5.0)** – піднесення дійсного числа **2.0** до степеня **5**. Зверніть увагу: вираз **pow(2, 5)** із двома цілими аргументами є помилковим.

3. Значенням **log10(2.0)** є (наближено) 0.30103, значенням **log(1)** – дійсне 0.

4. Значенням **fabs(-2.0)** є дійсне **2.0**, значенням **abs(-2)** – ціле **2**.

У стандарті мови C++ відсутні математичні константи, зокрема ті, що позначають числа π=3.141593… та *e* =2.7182818….

Натомість 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 4**. 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;system("pause"); return 0;**

**}**

Бібліотеки систем програмування мовою С++ містять різноманітні константи й численні підпрограми, що реалізують математичні та інші функції. Зауважимо: склад бібліотек у різних середовищах може бути різним, тому вичерпну інформацію про вміст бібліотек може дати лише довідка в конкретному середовищі або самі бібліотечні файли. У деяких версіях мови C++ ці константи замінено відповідними бібліотечними функціями.

**Функція** в мові **С++** – це частина програми, оформлена спеціальним чином. Якщо програма описує дії з розв'язання деякої задачі, то функція описує дії з розв'язання деякої частини цієї задачі, тобто підзадачі.

**Математичні функції**

Якщо треба використовувати у програмі математичні функції, слід долучити бібліотеку, яка містить ці функції, тобто увести директиву **#include <cmath>**

|  |  |
| --- | --- |
| **acos** | арккосинус |
| **asin** | арксинус |
| **atan** | арктангенс |
| **ceil** | округлення до найближчого більшого цілого числа |
| **cos** | косинус |
| **exp** | показникова функція |
| **abs (fabs)** | модуль цілого (дійсного)  числа |
| **log** | натуральний логарифм |
| **log10** | десятковий логарифм |
| **pow(x,y)** | вираховує значення x в степені у |
| **sin** | синус |
| **sqrt** | квадратний корінь |
| **tan** | тангенс |

**Правила запису математичних виразів**

* Кількість відкритих і закритих дужок у виразах повинна бути однаковою.
* Усі елементи виразів (дроби, показник степеня, індекси) записують у горизонтальному рядку.
* Вирази можна записувати у декількох рядках. "Розривати” вирази можна, наприклад, після символу арифметичної операції. Власне символ дублювати не потрібно.
* Операції ++ (інкремент) та -- (декремент) є унарними, тобто мають лише один операнд.
* Операція ++ додає одиницю до операнда, операція -- віднімає одиницю від операнда.

Обчислення в арифметичних виразах виконуються зліва направо згідно з  таким пріоритетом операцій:

1. стандартні функції, ++, --;

2. множення (\*), ділення (/), остача від ділення (%);

3. додавання (+) та віднімання (–).

Для здобуття правильного результату слід дотримуватися таких правил записування арифметичних виразів в операторах С++:

* кожна команда (інструкція) має завершуватись крапкою з комою (;)
* мова С++ є чутлива до регістру, тобто х та Х – це дві різні змінні;
* аргумент функції завжди записують у круглих дужках;
* знаки множення не можна пропускати (3ab –> 3\*a\*b);
* якщо знаменник або чисельник має операції (+, −, \*, /), то його слід записувати у круглих дужках;
* для записування раціональних дробів, у чисельнику або знаменнику яких є числові константи, хоча б одну з цих констант слід записати як дійсне число із зазначенням десяткової крапки, наприклад, 2⁄ 𝑘 записують як 2.0/k;
* радикали (тобто корінь кубічний і вище) замінюють на дробові степені, наприклад, кубічний корінь (𝑥 + 1) записують як  pow(x+1, 1/3.0);
* слід враховувати правила зведення типів, оскільки в арифметичних виразах можуть брати участь різнотипні дані та відбувається зведення типів.

**Приклади запису математичних виразів**



**Основи введення-виведення інформації**

Вирішення навіть найпростішої задачі на комп’ютері не обходиться без операцій введення-виведення інформації. Введення даних - це передача інформації ззовні в оперативну пам‘ять (далі – ОП) із зовнішнього носія; виведення даних - зворотний процес, коли дані після обробки передаються з ОП на зовнішній носій. Зовнішнім носієм може служити дисплей, друкований пристрій, гнучкий або жорсткий диск і т.ін. Передача даних програмі та виведення результатів програми є необхідним елементом програми.

Програми можуть отримувати вхідні дані декількома способами: із стандартного вхідного файлу, пов'язаного з клавіатурою, або із дискового файлу. Аналогічний поділ існує і при виведенні даних: результати роботи програми за замовчуванням направляються на екран дисплея (у стандартний вихідний файл), але можна перенаправляти їх і на інший носій інформації, наприклад, на дисковий файл, на контролер будь-якого периферійного пристрою комп'ютера тощо.

У сучасних мовах програмування основним поняттям введення та виведення даних є *потік* — послідовність символів або інших даних. У програмі потік зображує фізичний файл на зовнішньому носії даних (диску, клавіатурі або екрані монітора), тобто фізичний файл «видно в програмі» як потік даних. Операції обміну даних з файлом представлено в програмі як операції добування даних з потоку або дописування їх до нього. Cтандартним файлам введення та виведення у таких мовах відповідають стандартний потік введення (**cin** - в C++) та стандартний потік виведення (**cout** - в C++).

Алгоритмічні мови програмування використовують концепцію поелементного введення-виведення даних. Зазвичай, введення інформації з клавіатури супроводжується "ехо-сигналом" - на екрані дисплея з'являється зображення символів, що вводяться. В деяких системах програмування існують засоби, що дозволяють відключити таке ехо. Виведення даних організується, починаючи з позиції розміщення курсору. При цьому пробіли між даними, що виводяться, автоматично не вставляються - їх необхідно враховувати самим.

За допомогою відповідних засобів можна вводити або виводити дані тільки певних типів.

Для організації у мові С++ введення та виведення даних використовуються стандартні бібліотечні функції. Бібліотеки С++ підтримують два основних способи введення/виведення: потокове введення/виведення (заголовний файл fstream) та форматоване введення/виведення за допомогою функцій (заголовний файл stdio.h).

Потокове введення / виведення.

У мові C++ дії, що пов’язані з операціями введення і виведення, виконуються за допомогою функцій бібліотек. Функції ведення і виведення бібліотек мови дозволяють читати дані з файлів та пристроїв і писати дані у файли і на пристрої.

Бібліотека мови C++ підтримує три рівня введення-виведення даних:

* введення-виведення потоку;
* введення-виведення нижнього рівня;
* введення-виведення для консолі і порту.

При введенні-виведенні потоку всі дані розглядаються як потік окремих байтів. Для користувача потік — це файл на диску або фізичний пристрій, наприклад, дисплей чи клавіатура, або пристрій для друку, з якого чи на який направляється потік даних. Операції введення-виведення для потоку дозволяють обробляти дані різних розмірів і форматів від одиночного символу до великих структур даних..

За замовчуванням стандартні введення і виведення повідомлень про помилки відносяться до консолі користувача (клавіатури та екрана). Це означає, що завжди, коли програма очікує введення зі стандартного потоку, дані повинні надходити з клавіатури, а якщо програма виводить дані — то на екран.

Потокове виведення виконується за допомогою вихідного потоку cout та операції вставки у вихідний потік ”<<”. Стандартна операція ”<<” для операцій виведення перевизначається, тобто для неї визначаються нові можливості. Потокове введення аналогічне виведенню, але використовує перевизначену операцію вставки з вхідного потоку ”>>” для потоку введення cin.

Виконуючи операцію введення з клавіатури, комп’ютер тимчасово зупиняється і очікує на введення значення для змінної. У відповідь слід на клавіатурі набрати деяку послідовність символів, що зображує значення (ці символи з’являться на екрані). Уведені символи запам’ятовуються у буфері та передаються функціям введення тільки після натиснення клавіші Enter.

*Буфер* — це область пам’яті для тимчасового зберігання даних. Максимальний обсяг буфера становить 128 символів (байтів). Завдяки наявності буфера можливе редагування даних під час їх введення.

У мові C++ існує декілька бібліотек, які містять засоби введення-виведення, наприклад: **stdio.h, iostream.h.** Найчастіше застосовують потокове введення-виведення даних, операції якого включені до складу класів **istream** або **iostream**. Доступ до бібліотеки цих класів здійснюється за допомогою використання у програмі директиви компілятора **#include <iostream.h>** в пізніших версіях **#include <iostream>**.

**Для потокового введення даних** вказується операція «**>>**» («читати з»). Це перевантажена операція, визначена для всіх простих типів і покажчика на **char**. Стандартним потоком введення є **cin**.

Формат запису операції введення має вигляд:

**сin [>> values];**

де ***values*** — змінна.

Так, для введення значень змінних **х і у** можна записати:

**сin >> х >> у;**

Кожна операція «>>» передбачає введення одного значення. При такому введенні даних необхідно дотримуватись конкретних вимог:

* для послідовного введення декількох чисел їх слід розділяти символом пропуску (« ») або **Enter** (дані типу **char** роздiляти пропуском необов’язково);
* якщо послідовно вводиться символ і число (або навпаки), пропуск треба записувати тільки в тому випадку, коли символ (типу **char**) є цифрою;
* потік введення ігнорує пропуски;
* для введення великої кількості даних одним оператором їх можна розташовувати в декількох рядках (використовуючи Enter);
* операція введення з потоку припиняє свою роботу тоді, коли всі включені до нього змінні одержують значення. Наприклад, для операції введення **х і у**, що вказана вище, можна ввести значення **х та у** таким чином:

**2.345 789**  
або

**2.345**  
**789.**

Оскільки в цьому прикладі пропуск є роздільником між значеннями, що вводяться, то при введенні рядків, котрі мiстять пропуски у своєму складі, цей оператор не використовується. У такому випадку треба застосовувати функції **getline(), get()** тощо (це буде розглядатися окремо в подальших темах). У мові C++ бажано здійснювати потокові введення-виведення даних.

За стандартних налаштувань запис числа сприймається як десятковий. Якщо натиснути Enter, не набравши нічого, окрім пропусків, то комп’ютер і надалі чекатиме. В інструкції введення можна записати кілька імен змінних - кожне після «свого» знака ”>>”. При виконанні такої інструкції треба набрати на клавіатурі відповідну кількість вхідних констант, відокремивши їх одним або кількома порожніми символами.

**Для потокового виведення даних** необхідна операція **«<<»** («записати в»), що використовується разом з ім’ям вихідного потоку **cout**. Наприклад, вираз

**cout << х;**

означає виведення значення змінної **х** (або запис у потік). Ця операція вибирає необхідну функцію перетворення даних у потік байтів.

Формат запису операції виведення представляється як:

**cout << data [<< data1];**,

де **data, data1** — це змінні, константи, вирази тощо.

Потокова операція виведення може мати вигляд:

**cout << “у =” << x + а – sin(x) << “\n“;.**

Застосовуючи логічні операції, вирази треба брати в дужки:

**cout << “р =” << (а && b || с) << "\n";.**

Символ переведення на наступний рядок записується як рядкова константа, тобто **“\n”**, інакше він розглядається не як символ керуючої послідовності, а як число 10 (код символу).

Слід пам’ятати, що ***при виведенні даних з використанням «cout <<» не виконується автоматичний перехід на наступний рядок, для реалізації такого переходу застосовується так званий знак переведення рядка*** ***“\n” або операція*** ***endl.*** Тобто, вивести рядкову константу можна, наприклад, так:

**cout <<** **“**Bad news has wings **\n“;**

**або   cout <<** **“**Bad news has wings **” << endl;**.

Операція «помістити в потік» (”<<”) має асоціативність зліва направо:

(((cout<<"47+ 53 =")<<(47+53))<<endl)

Тобто, багаторазові операції ”<<” виконуються у тій послідовності, у якій вони записані. Такий спосіб об’єднання операцій ”<<” можливий, оскільки дана операція повертає посилання на об’єкт свого лівого операнда (на об’єкт cout). Отже, саме лівий вираз в круглих дужках (cout << "47 + 53 =")виводить заданий рядок символів і повертає посилання на cout*.* Тому середній вираз в круглих дужках має виконуватися, тобто виводити ціле значення 100 і повертає посилання на cout*,* наприклад: (cout<<(47+53)). Потім виконується самий правий вираз у круглих дужках, наприклад: cout << endl*,* який переводить рядок, скидає cout і повертає посилання на cout. Аналогічно, багаторазовою є і операції ”>>”.

Для керування потоковим введенням/виведенням можуть використовуватися маніпулятори потоків, які дозволяють міняти режими виконання операцій введення-виведення. Наприклад, маніпулятор **hex** задає форматування із шістнадцятковим перетворенням, **endl** - вставку символа нового рядка і очищення потоку.

Стандартні засоби потокового введення/виведення описані у бібліотеці iostream (iostream.h), маніпулятори потоків - у бібліотеці iomanip.h. Щоб мати змогу їх використати у програмі, відповідні бібліотеки слід підключити до програми за допомогою директиви #include. Наприклад,

#include <iostream.h>

int main()

{ double r,s;

const float pi=3.14;

cout<< "\nr="; cin >> r;

s=pi\*r\*r;

cout<< "\narea of a circle: " << s << endl;

system("pause");

return 0;

}

В C++ оператори об’єднуються в блоки — функції. **Функція** — це послідовність операторів. Кожна програма в C++ повинна містити **головну функцію** main(). Саме з першого оператора в main() і починається виконання програми. Функції, як правило, виконують конкретне завдання. Наприклад, функція max() може містити оператори, які визначають максимальне число з двох переданих їй. Функція calculateGrade() може вираховувати оцінку студента..

**Порада**: Пишіть функцію main() в файлі .cpp з тим же іменем, що має проект. Наприклад, якщо ви пишете програму Chess, то помістіть вашу функцію main() в chess.cpp.

У загальному випадку програма складається з декількох функцій, що не перетинаються (тобто **«вкладення» однієї функції в іншу неприпустиме**). Перед функціями і між ними можуть бути присутні оголошення об’єктів даних і оператори препроцесорної обробки. Функції користувача, які викликаються у головній функції main(), слід обов’язково описати до їх використання. Наведемо приклад запису фрагмента простої програми:

//..............фрагмент простої програми на С++

#include <iostream.h>

#include <соnio.h> //директиви препроцесора

const int n=20;

void main()

{

float mas[n]; //опис одновимірного масиву

//..............введення елементів масиву

for (int i=0; i < n; i++)

cin >> mas[i];

//......виконання перетворень

//......виведення перетвореного масиву

for(int i=0; i < n; i++)

cout << " " << mas[i] << " ";

getch (); // затримка результату на екрані

}

***Приклад 1.*** Написати програму, що містить організацію виведення даних, пояснювальні повідомлення, а також символи переведення рядка.

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int wozrast = 20;**

**int doplata = 2;**

**float zarplata = 309.75;**

**float prozent = 8.5;**

**cout << " Verification of source data \n";**

**cout << " Age"<< '\t' <<"additional payment "<< '\t' << "salary "<< '\t' << "percent:\n";**

**cout << wozrast << '\t' << doplata << '\t' << zarplata << '\t' << prozent << '\n';**

**}**

В останніх двох операціях виведення програми використано символи табуляції.

Для додаткового керування даними, що виводяться, використовують маніпулятори **setw(w)** та **setprecision(d)**. Маніпулятор **setw(w)**призначений для зазначення довжини поля, що виділяється для виведення даних (**w** — кількість позицій). Маніпулятор **setprecision(d)** визначає кількість позицій у дробовій частині дійсних чисел.

Маніпулятори змінюють вигляд деяких змінних в об’єкті **cout**, що у потоці розташовані за ними. Ці маніпулятори називають *прапорцями стану*. Коли об’єкт посилає дані на екран, він перевіряє прапорці, щоб довідатися, як виконати завдання, наприклад, запис:

**cout << 456 << 789 << 123;**

призводить до виведення значення у вигляді: **456789123**, що ускладнює визначення групи значень.

***Приклад 2.*** Написати програму, використовуючи маніпулятор **setw()**.

**#include <iostream.h>**

**#include <iomanip.h>**

**#include <conio.h>**

**main ( )**

**{**

**cout << 456 << 789 << 123 << endl;**

**cout << setw(5) << 456 << setw(5) << 789 << setw(5) << 123 << endl;**

**cout << setw(7) << 456 << setw(7) << 789 << setw(7) << 123 << endl;**

**getch ();**

**}**

Результати виконання програми:  
**456789123**   
**456     789     123**   
**456       789       123**

У цьому прикладі з’явився новий заголовний файл **iomanip.h**, що дозволяє застосовувати функції маніпуляторів. При використанні функції **setw()** число вирівнюється вправо в межах заданої ширини поля виведення. Якщо ширина недостатня, то вказане значення ігнорується.

Функція **setprecision(2)** повідомляє про те, що число з плаваючою крапкою виводиться з двома знаками після крапки з округленням дробової частини, наприклад, при виконанні операції

**cout** **<< setw(7) << setprecision(2) << 123.456789;**

буде отримано такий результат: **123.46.**

Функції **cout.width(w)** та **cout.precision(d)**, які потребують підключення тільки заголовного файла **iostream.h**, виконують дії, подібні тим, що і функції **setw(w)** та **setprecision(d)**.

Операція введення використовує ті ж самі маніпулятори, що й операція виведення.

При роботі з дійсними числами спочатку встановимо режим виведення з фіксованою крапкою, викристовуючи модификатор **std::fixed**. Потім, викристовуючи модификатор **std::setprecision(n)**, встанавлюємо точність виведення **n**. Для роботи модификатора **fixed** необхідно підключити заголовний файл **iomanip**. Приклад:

**#include <iomanip>**

**using namespace std;**

**//...**

**cout << fixed << setprecision(n) << something;**

***Приклад 3.*** Написати програму обчислення податку на продаж.

**#include <iostream.h>**

**#include <iomanip.h>**

**#include <conio.h>**

**main ( )**

**{ float prod\_sum;** *// prod\_sum — сума продаж*

**float nalog;**

cout << **"** Enter the amount of sales **";**

**cin >> prod\_sum;**

*//............... обчислення податку на продаж*

**nalog = prod\_sum\* 0.7;**

**cout << " " << setprecision(2) << prod\_sum;**

**cout << " " << setprecision(2) << nalog << "\n";**

**getch();** *// затримка екрана*

**return 0;**

**}**

Внаслідок того, що у першому операторі **cout** відсутня інструкція переведення рядка, відповідь користувача на підказку (тобто введене значення змінної **prod\_sum**) з’явиться відразу праворуч за самою підказкою.

**Коментарі** необхідні для пояснень призначення тих чи інших частин програми і їх текст завжди ігнорується компілятором. Мова С++ використовує два різновиди коментарів:

* *// текст* — **однорядковий коментар**, який починається з двох символів «/» («коса риска») і закінчується символом переходу на новий рядок;
* */\* текст \*/* — **багаторядковий коментар**, що розташовується між символами-дужками «/\*» і «\*/».

Багаторядкові коментарі не можуть бути вкладеними один в одний, а однорядкові коментарі можна вкладати в багаторядкові коментарі. Багаторядкові коментарі доцільно застосовувати для тимчасового виключення блоків при налагодженні програми.

Наведемо кілька порад стосовно раціонального складання коментарів:

* коментарі повинні бути добре складеними реченнями, мати правильну пунктуацію та містити тільки потрібну для супроводу інформацію;
* пропуск — один з найбільш ефективних коментарів, що значно поліпшує розуміння програми;
* штрихові лінії коментаря або порожні рядки застосовуються для поділу функцій та інших логічно завершених фрагментів програм.

Директива препроцесора **#include<iostream>** забезпечує підключення до програми засобів зв’язку зі стандартними потоками введення-виведення даних. Ці засоби знаходяться у заголовному файлі **iostream.h**, де **і (*input*)** — введення, **о (*output*)** виведення, **stream** — поток, **h (*head*)** — заголовок.

При створенні програми враховують такі основні вимоги:

* усі використані константи, змінні, функції та нестандартні типи повинні бути оголошеними (описаними) до їхнього першого використання, і ці оголошення можна розміщати в будь-якому місці програми;
* кожний оператор мови закінчується символом **«;»;**
* фігурні дужки (**« { »** та **« } »**) виділяють складений оператор і все, що подано між такими дужками, синтаксично сприймається як один оператор;
* вкладені блоки повинні мати відступ у 3-4 символи, при цьому блоки одного рівня вкладеності слід вирівняти за вертикаллю.

**Зовнішні оголошення**

*Зовнішні оголошення* – це глобальні описи даних (змінних або констант). Змінна у програмуванні є моделлю (зображенням) об’єкта в пам’яті комп’ютера. На фізичному рівні поняттю змінної відповідає група комірок оперативної пам'яті. Ці адреси ставляться у відповідність ідентифікаторам (іменам) змінних під час їх оголошення. Таким чином, ім'я змінної *вказує* (або *посилається*) на на першу комірку з групи, а величину групи визначає тип змінної (рис. 1).

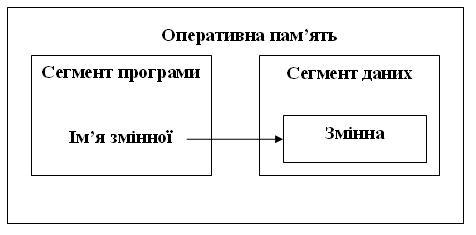
* 

Рис. 1. Ідентифікатор змінної та його асоціація з коміркою пам'яті

**Функція** – це частина програми, що виконує логічно завершений набір дій. В C/С++ вона є єдиним способом представлення підпрограм. Формат оголошення функції:

**тип\_результату ім'я([параметри]) *// заголовок функції***

**{ … } *// тіло функції***

Тіло функції може містити описи констант, змінних і оператори мови, які відокремлюються символом «;». Описи даних можуть зустрічатися в будь-якому місці функції, але до першого звертання до змінної. Якщо параметри у функції відсутні, то в заголовку все рівно записуються круглі дужки. Одна з функцій програми на мові C/С++ повинна мати ім'я main***.*** Саме з неї починається виконання програми. Функція main визначає дії, що виконуються програмою, і викликає інші функції. Якщо програма містить тільки одну функцію, то вона і є головною (має ім'я main).

Програма може містити довільне число директив препроцесора, оголошень і визначень змінних, описів функцій. Порядок появи цих елементів у програмі досить істотний: він впливає на можливість використання змінних, функцій і типів у різних частинах програми.

Логічні операції та операції порівняння в мові С/С++ оперують поняттями "істина" і "хиба"

**Областю дії оголошення змінної, тобто ідентифікатора змінної** є частина програми, в якій його можна використовувати для доступу до зв’язаної з ним області пам’яті. Залежно від області дії змінна може бути локальною або глобальною. Локальна змінна визначена всередині блока (блок розташований між фігурними дужками). Область її дії обмежена початком опису змінної та кінцем блока, включаючи усі вкладені блоки. Змінна, визначена поза будь-яким блоком, називається глобальною, і областю її дії вважається файл, у якому вона визначена від початку опису до його кінця.