**Лекція № 2. Основні поняття та типи даних мови програмування С++. Реалізація різних алгоритмічних структур на мові С++.**

1972 році співробітник фірми **Bell Laboratories** Деніс Рітчі створив нову алгоритмічну мову — С. В її основу було закладено багато особливостей мови **Assembler**. Мова С є універсальною, придатною для розв’язання будь-якого типу задач, хоча спочатку була задумана як мова системного програмування (у 1973 році на мові С Деніс Рітчі реалізував операційну систему **Unix**).

У середині 80-х років Б’ярн Страуструп розробив мову «С з класами», що надалі стали називати мовою C++. Ця мова дозволяє працювати не тільки зі змінними, але і з їх адресами, розміщувати дані як у пам’яті, так і в регістрах, використовувати непряму адресацію (завдання адреси комірки, в якій зазначена адреса даного), автоматично змінювати адресу. Об’єктний код, що формується компіляторами мови C++, займає приблизно стільки ж пам’яті, скільки і відповідна програма на **Assembler**.

Мову C++ можна розглядати як надмножину мови С, бо вона зберігає усі можливості, що надає мова С, і доповнює їх засобами об’єктно-орієнтованого програмування. C++ є універсальною алгоритмічною мовою, яка використовується для розробки системних та складних прикладних програм. Це не тільки найпоширеніша мова програмування, але й мова спілкування програмістів, оскільки більшість програм алгоритмів написані на C++.

Мова C++ є мовою високого рівня і основою багатьох систем програмування: **Borland C++**, **Visual C++**, **Borland C++ Builder**. Найбільш популярною з них вважається **Borland C++ Builder**. За допомогою цієї системи візуального об’єктно-орієнтованого програмування як користувач-початківець, так і програміст-професіонал мають можливість створювати інтерфейс користувача до прикладних програм різноманітних класів, що виглядає однаково професійно.

У природній мові спілкування виділяють чотири основні елементи: символ, слово, словосполучення та речення. Подібні елементи існують і в алгоритмічній мові, тільки слова мають назву *лексеми*, словосполучення — *вирази*, а речення — *оператори*. Лексеми створюються із символів, вирази — із лексем та символів, оператори — з символів, виразів і лексем.

**Алфавіт** мови C++ включає:

* великі (**A-Z**) і малі (**a—z**) літери латинського алфавіту та символ підкреслення ( **\_** );
* арабські цифри від **0** до **9**;
* знаки арифметичних дій **+, -, \*, /, %, ++, —**;
* знаки побітових операцій **<<, >>, &, |, ~, ^**;
* знаки відношень **<, <=, ==, !=, >, >=**;
* знаки логічних операцій **&&, ||, !**;
* розділові знаки **, ; : пропуск**;
* спеціальні знаки **., =, ->, ?, \, $, #, ‘, “**;
* символи дужок **(, ), [, ], {, }**.

Інші символи, а також літери кирилиці не використовуються для побудови базових елементів мови або для їх розділу, але вони можуть застосовуватись у символьних константах та коментарях.

**Лексеми**, тобто базові елементи мови з певним самостійним значенням, складаються із символів алфавіту. До них відносять

* ідентифікатори,
* ключові слова,
* знаки операцій,
* константи,
* роздільники (дужки, крапка, кома, символи пропуску).

Межі лексем визначаються іншими лексемами-роздільниками або знаками операцій.

***Ідентифікатором***, тобто ім’ям програмного об’єкта, називається будь-яка послідовність літер латинського алфавіту, цифр i символу підкреслення за умови, що першою стоїть літера або символ підкреслення, а не цифра.

Існує два різновиди ідентифікаторів:

* *стандартні*, наприклад, імена всіх вбудованих у мову функцій;
* *користувальницькі*.

Мова C++ чуттєва до регістру літер, тому компілятор розпізнає великі і малі літери латинського алфавіту як різні символи. Це дає можливість створювати ідентифікатори, що однаково читаються, але відрізняються написом одного або декількох символів. Наприклад, ідентифікатори **«Sigma»**, **«sigma»** і **«sigMa»** вважаються різними.

Ідентифікатори можуть мати будь-яку довжину, але значимими є не більше 31 символу від початку ідентифікатора, a в деяких компіляторах це обмеження ще більш суворе (не більше 8 символів). Імена програмних об’єктів створюються на етапі оголошення даних, після цього їх можна використовувати в різних операторах програми.

**Рекомендації щодо вибору ідентифікатора**:

* рекомендується не жалкувати часу на створення ідентифікаторів переважно за їх змістовим призначенням;
* ім’я програмного об’єкта повинно легко розпізнаватись і, бажано, не мати символів, які можна переплутати між собою;
* ідентифікатор не повинен збігатися з ключовими словами, а також з іменами стандартних об’єктів мови C++;
* не слід починати ідентифікатори із символу підкреслення, бо вони можуть співпадати з іменами системних функцій або змінних;
* для розділу частин імені можна використовувати символ підкреслення;
* всередині ідентифікатора не можна розміщати символи пропуску.

***Ключовими*** (службовими) словами називають ряд зарезервованих ідентифікаторів, що вживаються для побудови конструкцій мови і мають фіксоване значення. За смисловим навантаженням службові слова поділяються на такі основні групи:

* специфікатори типів — **char, int, long, typedef, short, float, double, enum, struct, union, signed, unsigned, void;**
* квалифікатори типів — **const і volatile;**
* класи пам’яті — **auto, extern, register, static;**
* для побудови операторів — **for, while, do, if, else, switch, case, continue, goto, break, return, default, sizeof.**

Як роздільники лексем застосовуються такі символи: пропуск, табуляція, символ нового рядка, коментар. Між будь-якими двома лексемами допускається довільна кількість символів-роздільників. Крім того, деякі лексеми (**«\*», «+», «,», « », «(», «->»** тощо) самі є роздільниками і відділяти їх від інших лексем символами-роздільниками необов’язково.

Таблиця 1. Список ключових слів мови С++



**Спеціальні символи**

|  |  |
| --- | --- |
| Символ | Опис |
| **\n** | Новий рядок (LF) |
| **\t** | Горизонтальна табуляція (Н\_TAB) |
| **\r** | Повернення каретки (CR) |
| **\f** | Нова сторінка (FF) |
| **\a** | Звуковий сигнал |
| **\'** | Апостроф |
| **\”** | Лапки |
| **\\** | Обернена коса риска (back slash) |

**Структура програми**

Основними частинами типової структури програми на С++ є такі:

директиви препроцесорної обробки;

опис зовнішніх змінних (вихідних даних і результатів) та функцій;

функції програми;

головна функція — програми **main()**, що має вигляд:

main()

{

опис змінних; виконавчі оператори;

}

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. Приклад простої програми:

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

**#include <iostream.h>**

**#include <соnio.h> //директиви препроцесора**

**const int n=20;**

**int main()**

**{**

**float mas[n]; //опис одновимірного масиву**

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

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

**cin >> mas[i];**

**// виконання перетворень та виведення перетвореного масиву**

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

**{ mas[i]= mas[i]\*2;**

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

**getch (); // затримка результату на екрані**

**return 0;**

**}**

**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 вкладатися один в одний, а однорядкові коментарі можна вкладати в багаторядкові коментарі. Багаторядкові коментарі доцільно застосовувати для тимчасового виключення блоків при налагодженні програми.

Рекомендації щодо 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.

У другому рядку записано директиву препроцесора.

***Препроцесор*** – складова частина компілятора, яка проводить, попередню обробку програми. Директиви записують в окремих рядках і починають символом #. Директива *#include <iostream>* и під’єднує бібліотечний файл iostream.h (де **і (*input*)** — введення, **о (*output*)** виведення, **stream** — поток, **h (*head*)** — заголовок). Саме у цьому файлі описані функції, які дають змогу виконувати операції введення-виведення даних (ім‘я cout, операцію << та багато інших). Слово include (включити) означає, що препроцесор перед компіляцією програми має включити в неї вміст спеціального файлу зі складу системи програмування, ім‘я якого записано в кутових дужках *<iostream>*. Без включення цього файлу ім‘я cout буде невизначеним, компілятор повідомить про цю помилку. Файл iostream є одним із багатьох заголовних (header) файлів (або h-файлів), що входять до складу системи програмування, тобто є стандартними.

Директиви мови С++ розпочинаються символом #

**#include** – директива підключення файлу, бібліотеки

Директива #define

#define –має два значення:

1 – оголошення константи

         #define N 25

2- описати макроси (короткі команди, функції тощо.)

         #define D(a,b,c) ((b\*b)-4\*(a)\*(c))

         #undef D

         #define D(a,b,c) ((a)\*(a)\*(a))

*using namespace std;*   -  інструкція компілятору «використати простір імен std». Цей простір імен стандартним. Завдяки наведеній інструкції спрощується доступ до бібліотечних засобів (один з них з ім‘ям cout, використовується в програмі.

Далі у програмі записана обов’язкова функція *mаіn*(). Ключове слово int означає, що функція main() повертатиме у точку виклику результат цілого типу.

Конструкція cout « забезпечує виведення на екран монітора повідомлення.

Команда return слугує для виходу з функції main(). Числовий параметр після return є результатом (значенням) функції (у цій програмі - 0).

Функція з ім‘ям main називається головною функцією. Вона має бути в кожній програмі, з неї починається виконання програми й зазвичай нею закінчується. Ім‘я main не є зарезервованим, але не рекомендується його використовувати в інших цілях.

Вміст подальших рядків утворює тіло функції, яке починається символом {, і закінчується символом }. У тілі функції задано дії у вигляді послідовності інструкцій. Інструкція cout<<"Hello world!"; задає виведення на екран повідомлення.

Воно з'являється у вікні програми, яке має відкритися на екрані під час її виконання і зникнути після завершення. Текстове повідомлення , що виводиться на екран, записується в лапках "". Команда return слугує для виходу з функції main(). Числовий параметр після return є результатом (значенням) функції (у цій програмі - 0).

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, але для кращого сприйняття змісту краще робити це на початку;
* кожний оператор мови закінчується символом **«;»;**
* фігурні дужки (**« { »** та **« } »**) виділяють складений оператор і все, що подано між такими дужками, синтаксично сприймається як один оператор;
* для кращого сприйняття змісту вкладені блоки повинні мати відступ у 3-4 символи, при цьому блоки одного рівня вкладеності слід вирівняти за вертикаллю.

# Типи даних

Обробка даних різного типу є головною метою будь-якої програми. Кожне з даних характеризується класом пам’яті, ім’ям, типом і значенням. *Імена* дозволяють ідентифікувати дані, тобто відрізняти їх між собою. Програміст обирає тип кожної величини, що використовується для подання реальних об’єктів. *Тип* задає множину можливих значень даних і способи їх зберігання, перетворення та використання.

**Обов’язкове оголошення типу даних дозволяє** компілятору робити перевірку допустимості різних конструкцій програми.

Усі типи даних мови C++ можна розділити на **основні** (базові) і **складені**. Основні типи визначені для представлення цілих, дійсних, символьних і логічних даних. На основі цих типів вводиться опис складених типів, до яких належать масиви, перелічення, функції, структури, посилання, покажчики, об’єднання і класи.

**Основні типи даних** (*див. табл. 2, 3*) часто називають арифметичними, тому що їх можна використовувати в арифметичних операціях. Для опису основних типів мови C++ використовують такі службові слова:

* **int** (цілий);
* **char** (символьний);
* **bool** (логічний);
* **float** (дійсний);
* **double** (дійсний з подвійною точністю);
* **void** (порожній, не має значення).

Типи **int, char, bool** називають ***цілими***, а типи **float** та **double** — ***дійсними з плаваючою крапкою***. Код, що формує компілятор для обробки цілих величин, відрізняється від коду для величин з плаваючою крапкою.

Для уточнення внутрішнього подання та діапазону значень стандартних типів **мова C++ використовує чотири специфікатори типу**:

* **short** (короткий);
* **long** (довгий);
* **signed** (знаковий);
* **unsigned** (беззнаковий).

Таблиця 2. Дані цілого типу

|  |  |  |  |
| --- | --- | --- | --- |
| *Тип даних* | *Розмір,*  *байт* | *Нижня межа діапазону* | *Верхня межа діапазону* |
| [signed] char | 1 | -128 | 127 |
| unsigned char | 1 | 0 | 255 |
| [signed] short | 2 | -32 768 | 32 767 |
| unsigned short | 2 | 0 | 65 535 |
| [signed] int | 2 | -32 768 | 32 767 |
| 4 | -2 147 483 648 | 2 147 483 647 |
| unsigned int | 2 | 0 | 65 535 |
|  | 4 | 0 | 4 294 967 295 |
| [signed] long | 4 | -2 147 483 648 | 2 147 483 647 |
| unsigned long | 4 | 0 | 4 294 967 295 |

Таблиця 3. Дійсні типи даних

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Тип даних* | *Розмір,байт* | *Точність* | *Нижня межа діапазону* | *Верхня межа діапазону* |
| float | 4 | 1 | 3.4Е-38 | 3.4Е+38 |
| double | 8 | 15 | 1.7Е-308 | 1.7Е+308 |
| long double | 10 | 15 | 3.4Е-4932 | 1.1Е+4932 |

Розмір однакового типу даних може відрізнятися на комп’ютерах різних платформ, а також може залежати від застосованої операційної системи. Тому при оголошенні тієї чи іншої змінної потрібно чітко уявляти, скільки байт вона буде займати в пам’яті ЕОМ, щоб запобігти проблемам, пов’язаним з переповненням і неправильною інтерпретацією даних. Діапазони кожного з типів повинні бути перевірені для конкретного комп’ютера.

Цілі та дійсні типи разом називають арифметичними.

# Змінні

Кожна програма потребує виконання різноманітних обчислень, для здійснення яких використовуються вирази, що складаються з операндів, знаків операцій і дужок. Операнди задають дані для обчислень, а операції задають дії, які потрібно виконати над цими даними. Операнд є виразом, що в окремому випадку може бути константою або змінною.

**Змінна** — це іменована область пам’яті, у якій зберігаються дані визначеного типу. Змінна має ім’я, розмір та інші атрибути, такі як видимість, час існування тощо. *Ім’я* *змінної* служить для звертання до області пам’яті, у якій зберігається її значення. Кожна змінна повинна мати своє ім’я, причому в одному блоці не може бути двох змінних з однаковим ім’ям. ***Перед використанням будь-яка змінна повинна бути описана***, при цьому для неї резервується деяка область пам’яті, розмір якої залежить від конкретного типу змінної. Під час виконання програми змінна може приймати різні значення. Загальний вигляд опису змінних:

**[клас пам’яті] [const] тип ім’я [ініціювання];**

де необов’язковий *клас пам’яті* може приймати одне зі значень — **auto, extern, static** чи **register** (тут і далі при описі синтаксису об’єктів програмування необов’язкові частини синтаксичних конструкцій мови подано у квадратних дужках «[]»);

модифікатор **const** вказує, що змінна не може змінювати своє значення, у цьому випадку її називають **типізованою (іменованою) константою** або просто **константою**;

**ініціювання** — це присвоювання змінній при описі початкового значення, яке записується зі знаком рівності — **= значення** або в круглих дужках — (**значення**). Константа ***повинна бути ініційована при описі***. Один оператор може містити опис декількох змінних одного типу, розділяючи їх комами, наприклад:

**const int n = 20, m = 5, k = 4;** — ініціювання констант **n, m, k** цілого типу;

**float h = 17.5, d(5.5), sum**; — опис дійсних змінних **h, d, sum**, ініціювання **h і d**;

**char sf = 'f', st[ ] = "Knowledge is power."**; — ініціювання символьних змінних.

Якщо **тип** значення, що ініціюється, не збігається з типом змінної, то виконуються перетворення типу.

**Областю дії 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.

**Клас пам’яті** визначає час існування та область видимості програмного об’єкта, тобто змінної. Якщо клас пам’яті не зазначений явно, то він визначається компілятором, виходячи, з контексту оголошення.

***Час існування*** змінної може бути постійним (протягом виконання програми) і тимчасовим (протягом виконання блока).

***Областю видимості*** ідентифікатора називається частина тексту програми, з якої можна здійснити звичайний доступ до зв’язаної з ідентифікатором області пам’яті. Найчастіше область видимості збігається з областю дії. Винятком є ситуація, коли у вкладеному блоці описана змінна з таким же ім’ям. У цьому випадку зовнішня змінна у вкладеному блоці невидима, хоча він і входить до її області дії. Проте до цієї змінної, якщо вона глобальна, можна звернутися, застосовуючи операцію доступу до області видимості — **“::“**.

Клас пам’яті задають такі специфікатори:

* **auto** — автоматична змінна, для якої пам’ять виділяється у стеку і за необхідності ініціюється кожного разу при виконанні оператора, що містить її визначення. Звільнення пам’яті відбувається при виході з блока, де описана змінна. Час її існування — з моменту опису до кінця виконання блока. Для глобальних змінних цей специфікатор не використовується, а для локальних він приймається за замовчуванням, тому можна не задавати явно;
* **extern** означає, що змінна визначена в іншому місці програми (в іншому файлі або далі по тексту) і використовується для створення змінних, доступних в усіх модулях програми, де вони оголошені. При ініціюванні змінної у тому ж операторі, спеціфікатор **extern** ігнорується;
* **static** — статична змінна, що має постійний час існування. Вона ініціюється один раз при першому виконанні оператора, що містить визначення змінної. Залежно від розташування оператора, описані статичні змінні можуть бути глобальними і локальними. Глобальні статичні змінні видимі тільки у тому модулі, в якому вони описані;
* **register** — аналогічний до специфікатора **auto**, але пам’ять виділяється по можливості в регістрах процесора і за відсутності такої можливості у компілятора змінні обробляються як **auto**.

Приклад фрагменту програми з використанням розглянутих вище понять:

**int d;** //1 — глобальна змінна **d**

**int main()**

**{**

**int b;** //2 — локальна змінна **b**

**extern int y;** //3 — змінна **у** визначена в іншому місці програми

**static int s;** //4 — локальна статична змінна **s**

**d = 1;** //5 — присвоювання значення глобальній змінній

**int d;**   //6 — локальна змінна **d**

**d = 10;** //7 — присвоювання значення локальній змінній

**::d = 3;**  //8 — присвоювання значення глобальній змінній

**return 0;**

**}**

**int у = 4;**       // 9 — визначення та ініціювання змінної **у**

У цьому прикладі глобальна змінна **d** визначена поза всіма блоками. Пам’ять для неї виділяється в сегменті даних на початку роботи програми, областю дії є вся програма. Область видимості — вся програма, крім рядків 6-8, тому що в першому з них визначається локальна змінна з тим же ім’ям, область дії якої починається з початку її опису і закінчується при виході з блока. Змінні **b i s** — локальні, область їх видимості — блок, але час існування різний: пам’ять під **b** виділяється в стеку при вході у блок і звільняється при виході з нього, а змінна **s** розташована у сегменті даних та існує увесь час роботи програми. Якщо початкове значення змінних явно не задається, компілятор присвоює глобальним і статичним змінним нульове значення відповідного типу. Автоматичні змінні не ініціюються. ***Початкове ініціювання змінних не є обов’язковим, проте все ж його бажано здійснювати***.

Опис змінної може виконуватися у формі оголошення або визначення. Оголошення інформує компілятор про тип змінної і класи пам’яті, а **визначення** містить, крім цього, вказівку компілятору про виділення пам’яті відповідно до типу змінної. У C++ більшість оголошень є одночасно і визначеннями (у наведеному вище програмному фрагменті тільки опис **extern int у**; є оголошенням, але не визначенням). ***Змінна може бути оголошена багаторазово, а визначена тільки в одному місці програми***, оскільки оголошення тільки описує властивості змінної, а визначення зв’язує її з конкретною областю пам’яті.

**Основні типи змінних.**

***Цілі змінні*** (типу **int, long, short**) необхідні для збереження цілих значень і можуть бути знаковими і беззнаковими. Знакові змінні застосовують для подання як додатних, так і від’ємних чисел, при цьому один біт (найстарший) виділяється під знак. Для оголошення беззнакової змінної, тобто змінної, що приймає тільки додатні значення, необхідно використовувати ключове слово **unsigned**. За замовчуванням будь-який цілий тип вважається знаковим, і тому немає потреби у використанні ключового слова **signed**.

***Символьний тип*** даних **char** застосовується у випадку, коли змінна містить інформацію про код ASCII або для побудови таких більш складних конструкцій, як рядки, символьні масиви тощо. Дані типу **char** також можуть бути знаковими і беззнаковими.

Символьний тип — це множина символів кодової таблиці комп'ютера ASCII. Символьна стала - це один символ, узятий у лапки на зразок апострофа, або число у 8-, 10- чи 16-й системі числення, яке є кодом символу у таблиці ASCII.

Приклад . Розглянемо описи символьних змінних, де змінним ml, m2, m3 і m4,яким надамо значення латинської літери "А" чотирма способами:

char ml = ’А', m2 = 0101, m3 = 65, m4 = 0x41;

Число 65 - це десятковий код символу 'А', 101 - вісімковий, 41 - шістнадцятковий. На початку останніх двох кодів (101, 41) записують префікси "0” чи "0х" відповідно.

***Змінна типу* bool** займає 1 байт і використовується, насамперед, у логічних операціях, тому що може приймати значення **0** (**false** — «неправда») або відмінне від нуля (**true** — «істина»). У випадку перетворення до цілого типу **true** має значення 1.

Зауважимо, що не всі компілятори підтримують тип даних bool. Тому, перед тим як його використовувати, варто з’ясувати можливості компілятора.

Стандарт C++ визначає три типи даних для збереження ***дійсних значень змінних***: **float, double** та **long double** (типи ***з плаваючою крапкою***). Тип **float**, як правило, використовують для збереження не дуже великих дробових чисел.

***Змінна типу* void** не має значення, оскільки множина значень цього типу порожня. Такі змінні необхідні для узгодження синтаксису. Тип **void** використовується для визначення функцій, що не повертають значення, для вказівки порожнього списку аргументів функції, а також як базовий тип для покажчиків i в операції приведення типів. Наприклад, якщо немає потреби у використанні поверненого значення функції, перед ім’ям функції ставиться тип **void**:

**void minmax(int\*x, int k, int\*min, int&max);**.

# Константи

Константи являють собою фіксовані значення, що не можуть змінюватися впродовж виконання всієї програми.

Спосіб визначення кожної константи залежить від її типу. Константи мови С++ слід поділяти на літеральні та типізовані.

**Літеральна константа** — це лексема, що являє собою зображення фіксованого числового, рядкового або символьного значення. Такі константи бувають **цілі, дійсні, символьні та рядкові**

|  | Літеральні константи мови |  |
| --- | --- | --- |
| Константа | Формат | Приклади |
| Цiла | Десятковий: послідовність десяткових цифр  (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), що починається не з нуля, якщо це не число нуль | 9, 0, 217925 |
|  | Вісімковий: нуль, за яким розташовані вісімкові цифри (0, 1, 2, 3, 4, 5, 6, 7) | 02, 050, 07245 |
|  | Шістнадцятковий: 0х чи 0Х, за яким розташовані шістнадцяткові цифри (0, 2, 3, 4, 5, 6, 7, 8, 9, А, В, С, D, Е, F) | 0x1B9, 0X00FF |
| Дійсна | Десятковий:[цифри].[цифри]  Експоненціальний: [цифри][.][цифри]{Е|е}[+|-][цифри] | 9.7, .001, 87. 0.7Е6, .15е-3 9.2, 920 е-2, 92.Е-1, .92Е1 |
| Символьна | Один чи два символи, що подаються в апострофах | ‘А’ , ‘ю’ , ‘\* ‘ , ‘db’ , ’\0′, ‘\n’, ’\012′, ‘\х07\х07’ |
| Рядкова | Послідовність символів, що подають­ся в лапках | “RESULT”, |

**Цілі константи** можуть бути десятковими, вісімковими та шiстнадцятковими.

Довгі цілі константи (**long**) мають літеру **l** або **L** в кінці, наприклад: 32768L; 0777777l; 0XFL. Для завдання константи, без знака (**unsigned**) застосовується літера **u (U)**,наприклад 65535u. Довгі константи без знака записуються з використанням двох літер відразу: **(ul, UL)** або **(lu, LU)**.

**Дійсні числа**у мовах програмування мають дві форми подання: десяткову (природну) та експоненціальну (показникову).

***Десяткова форма***дійсного числа — це звичайний десятковий формат запису дійсного числа, тільки частина дійсного числа відділяється від дробової крапкою, а не комою, наприклад 10.123, 1.0123, 1012.3, 0.0010123.

**Експоненціальна форма**дійсного числа використовується для запису дуже великих або дуже малих чисел, для яких задавати зайві нулі не зовсім зручно, наприклад: 1 .0123\*1020,1.0123\*10-10. У цій формі запису числа можна виділити такі основні характеристики: знак числа, мантису числа, знак порядку та порядок числа. Зазначені характеристики дійсного числа зберігаються у пам’яті комп’ютера. Число у показниковій формі може бути представлено, наприклад, так: 1.0123Е-10. Мантиса записується ліворуч від знаку експоненти(***Е чи е***), порядок — праворуч. Символ **Е (е)** означає основу ступеня 10, і компілятор розпізнає цей запис як форму представлення дійсного числа. Символ пропуску всередині числа не допускається, а для відділення цілої частини від дробової використовується не кома, а крапка. При додатних значеннях числа і мантиси знак «+» можна не вказувати.

Як десяткова, так і екcпоненціальна форми запису допускають відсутність або цілої частини, або дробової, але не двох одразу.

За замовчуванням всі дійсні константи мають тип **double** — подвійну точність, що найчастіше займає в пам’яті 64 біти, тобто 8 байтів. Але у випадку, якщо програміста не влаштовує тип за замовчуванням, його можна вказати явно за допомогою спеціальних літер. Так, додавши літеру **f чи F**, константі надають дійсний тип **float** зі звичайною точністю, наприклад, 8.5f. Якщо в представленні константи використовується літера **L чи l**, то вона має тип **long double**.

Зображення від’ємної цілої чи дійсної константи вважається константним виразом, що складається зі знака унарної операції зміни знака (-) та константи, наприклад: -273, -2730.е-1, -273L.

***Символьні константи***мають один або два символи, що подаються в апострофах. Односимвольні константи займають у пам’яті один байт і мають стандартний тип **char** (***character-символ***). Двосимвольні константи займають два байти і мають тип **int**. Символьні константи мають цілий тип і їх можна ви­користовувати як цілочислові операнди у виразах.

**Послідовності.**

Послідовності починаються зі знака «\», їх називають **керуючими**або **escape-послідовностями.**Символ зворотної косої риски «\» використовується для запису кодів,що не мають графічного зображення, для запису символiв, а також для виведення символьних констант, якщо їх коди заданi у вісімковому та шістнадцятковому вигляді (*табл. 4*).

*Таблиця 4* **Керуючі послідовності мови С++**

|  |  |
| --- | --- |
| \а | звуковий сигнал |
| \b | повернення на крок |
| \f | переведення сторінки (формату) |
| \n | новий рядок |
| \r | повернення каретки |
| \t | горизонтальна табуляція |
| \v | вертикальна табуляція |
| \\ | символ \ — зворотна коса риска |
| \’ | символ ‘ — апостроф |
| \” | символ ” — лапки |
| \0 | нуль-символ |
| \? | знак питання |
| \0ddd | вісімковий код символу |
| \0xddd | шістнадцятковий код символу |

**Рядкова константа *(рядковий літерал)***— це послідовність cимволів, що подається в лапках (тобто в символах **«”»**)і зберігається у неперервній ділянці пам’яті, наприклад: **“**As busy as a bee**”**. У кінець кожного рядкового літералу компілятором додається нуль-символ, що предствляється керуючою послі­довністю **«\0».** Тому довжина рядка завжди на одиницю більше кількості символів у його записі. Таким чином, порожній рядок (” “) має довжину 1 байт. Слід звернути увагу на різницю між рядком з одного символу, наприклад, “С” і символьною конс­тантою ‘С’ . Порожня символьна константа неприпустима.

Керуючі послідовності можуть також застосовуватись у рядкових константах. Так, якщо всередині рядка потрібно записати лапки, то перед ними слід розташувати зворотну косу риску («\»), за якою компілятор відрізняє їх від лапок, що обмежують рядок:

Рядки,що записані у програмі підряд або через символи пропуску, при компіляції конкатенуються («склеюються»). Тобто послідовність двох рядків

**“** **As the tree, so the fruit** **”**

**“** **- Яблуко від яблуні недалеко падає. ”**

цілком еквівалентна рядку:

**“ As the tree, so the fruit - Яблуко від яблуні недалеко падає. ”.**

Довгу рядкову константу можна розмістити також на декількох рядках. У цьому випадку ставиться зворотна коса риска і натискається клавіша **Enter**. Наприклад:

**“Програма виконує те, \**

**що ви їй наказали робити, а не те, \**

**що ви хотіли, щоб вона робила.”**

Існує інша можливість завдання констант — з використанням директиви препроцесора **#define**,при цьому оголошення має вигляд:

**#define ім’я константи значення константи**

і наприкінці такого запису символ **«;»** не ставиться, тобто:

**#define max 65532**

**#define km 1000.**

Директива **#define** визначає ідентифікатор (*ім’я константи*) і послідовність символів (*значення константи*), яка замінює ідентифікатор у тексті програми.

**Нульовий покажчик** (*NULL-покажчик*)— єдина неарифметична константа мови C++.

При застосуванні великої кількості логічно взаємозалежних констант C++ доцільно користуватися константами перелічення. Тип перелічення має вигляди:

**enum {список іменованих констант}; — неiменоване перелічення,**

**enum [ім’я] {список іменованих констант}; — iменоване перелічення.**

де **enum** — службове слово (**enumerate** — перелічувати);

**ім’я** — ім’я списку констант;

**список іменованих констант** — розділена комами послідовнiсть iдентифікаторів або іменованих констант вигляду:

**ім’я константи = значення константи.**

Наприклад:

**enum {Anton, Ivan, Piter};**

**enum Months {January = 1, February, Marth, April, May, June, July, August, September, October, November, December};**

Якщо значення константи перелічення не визначено, то воно на одиницю більше значення попередньої константи. За замовчуванням перша константа має значення **0**. Тоді у першому прикладі константи одержать значення:**Anton = 0, Ivan = 1, Piter = 2**, а у другому — значення: J**anuary = 1, February = 2, Marth= З** тощо. Іменовані перелічення задають унікальний цілочисловий тип і можуть використовуватися як специфікації типу для визначення змінних.

# Операції

Для здійснення маніпуляцій з даними мова C++ застосовує широкий набір операцій (див. *табл. 5*), що виконують формування і, відповідно, подальше обчислення виразів. Вирази містять **одну** або **декілька** операцій, об’єкти яких називають операндами. Операції являють собою деяку дію, що виконується над **одним** (***унарна***) або **декількома** (***бінарна, триарна***) операндами, і мають позначення (наприклад, операція перевірки на рівність — позначення «==», операція обчислення залишку від ділення цілих чисел — позначення «%» тощо).

Операції поділяються на:

* ***унарні*** або ***одномісні*** — **&, \*, -, +, ~, !, ++, –-, sizeof;**
* ***бінарні*** або ***двомісні*** — **+, -, \*, /, %, <<,  >>, &, :, ^, <, >, <=, ==, >=, !=, &&, ||, =,\*=, /=, %=, +=, -=, <<=, >>=, &=, |=, ^=, ., ->, ,, (), [];**
* умовну ***триарну*** або ***тримісну*** операцію — **?: .**
* **Основні операції мови C++**

|  |  |  |
| --- | --- | --- |
| № | Операції | Порядок виконання |
| 1 | ()  ,  {}  ->  . | Л -> П |
| 2 | !   ~   ++  —  &  \*  (type) | П -> Л |
| 3 | sizeof | П -> Л |
| 4 | \*   /   % | Л -> П |
| 5 | +   – | Л -> П |
| 6 | <<   >> | Л -> П |
| 7 | <   <=   >   >= | Л -> П |
| 8 | ==   != | Л -> П |
| 9 | & | Л -> П |
| 10 | ^ | Л -> П |
| 11 | | | Л -> П |
| 12 | && | Л -> П |
| 13 | || | Л -> П |
| 14 | ?: | П -> Л |
| 15 | =   +=   \*=   -=   /=   %= | П -> Л |
| 16 | , | Л -> П |

Порядок застосування операції визначається **пріоритетом операції** (яка операція виконується раніше, а яка пізніше) та **асоціативністю** (виконується зліва направо або справа на ліво). У першу чергу реалізуються операції з найвищим пріо­ритетом.

У табл.5 літерою **«Л»** позначено величину, що стоїть ліворуч від знака операції, літерою **«П»** — величину, яка розташована праворуч від знака операції, а символом «->» напрямок виконання операції. Розглянемо основні операції.

***Арифметичні операції:***

**+** — додає величину **П** до **Л**;

* + — віднімає **П** із **Л**;
  + — унарна операція зміни знака величини **П**;

**\***     — множення **П** і **Л**;

**/**  — ділення **Л** на **П**;

**%**  — залишок від ділення величини **Л** на величину **П** (для цілих чисел), наприклад, якщо **int g = 12;**, то операція **g = g % 9;** надасть результат: **g = 3;**

**++** — унарна операція інкремент. Якщо змінна розташовується праворуч від знака операції (префіксна форма), то значення збільшується на 1 до використання. Якщо ж змінна знаходиться ліворуч від знака операції (постфіксна форма), то її значення збільшується на 1 після використання, наприклад:

**int d;**

**++d;** — префіксний інкремент,

**d++;** — постфіксний інкремент;

-- — унарна операція декремент аналогічно інкременту має двi форми: префіксную (змінна розташована праворуч від знака операцii) — зменшення значення змінної на 1 відбувається до її використання; постфіксну (змінна знаходиться ліворуч від знака операції) — зменшення значення змінної на 1 після її використання.

**Операції присвоювання:**

**=** — присвоювання значення **П** змінній **Л**;

**+=**  — додає величину **П** до змінної **Л**;

**–=** — віднімає величину **П** від змінної **Л**;

**\*=**  — множення змінної **Л** на величину **П**;

**/=** — ділення **Л** на **П**;

**%=** — видає залишок від ділення **Л** на **П**.

Просте присвоювання здійснює операція «=». Допускається одночасне зчіплювання декількох операцій присвоювання за умови, що всі операнди мають однаковий тип, наприклад:

int і, j, с;

i **= j = с = 0;**

Операції **«+=», «-=», «\*=», «/=»** виконують складні присвоювання і дозволяють записувати вирази коротше, наприклад:

**s += 7;**       //s=s + 7;

**і \*= j + 5;**   //i=i\*(j +5);

g%=9;        //g=g%9;.

**Операції відношення** / ***порівняння*** порівнюють значення **Л** зі значенням **П**:

|  | **операція** | **відповідник** | **приклади** |
| --- | --- | --- | --- |
| Менше  Менше або дорівнює  Більше  Більше або дорівнює | <  <=  >  >= | <  ≤  >  ≥ | a < b  a<=b  a>b  a>=b |
| Дорівнює  Не дорівнює | ==  != | =  ≠ | a==b  a!=b |

Порівнювати можна операнди будь-якого типу, але вони повинні бути того ж самого вбудованого типу даних (порівняння на рівність і нерівність працює для двох величин будь-якого однакового типу даних), або між ними повинна бути визначена відповідна операція порівняння. Результат – логічне значення true або false. Використовуються для порівняння значень і дають відповідь false (0) true (1).

Операнди можуть бути символи, логічні, числові, покажчики.

Умова, яка перевіряється, може містити значення, яке не є логічним. Тоді значення нуль (0) сприймається як false, а значення  ≠0 - як true. Наприклад:

**int a = 10;**

**int b = 4;**

**int c = a == b; // 0**

**int c = 10 <= 4; // 0**

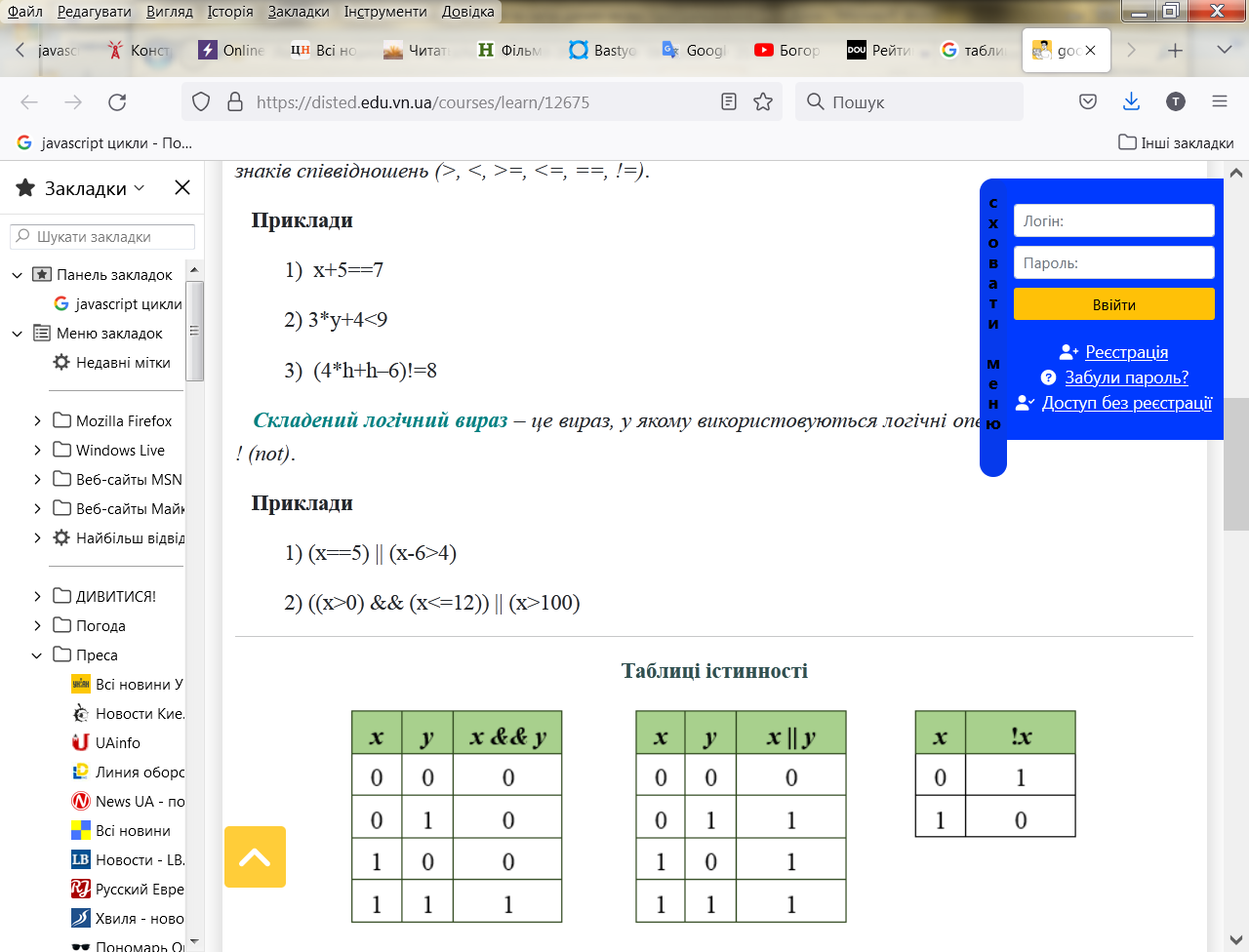
**Логічні операції** оперують з цілими значеннями або зі значеннями, які можна перетворити на цілі. Обчислення зупиняється, які тільки визначиться, чи є вираз правдивим («істина») або помилковим («неправда»). При цьому, як і для операцій відношення, значенням «істина» відповідає 1, а значенням «неправда» — 0.

Логічні операції реалізують операції математичної логіки. Серед логічних операцій є унарні й бінарні.. Дають відповідь типу bool:  false (0)  та true (1).

| **Операція** | **Відповідник** | **Приклади** |
| --- | --- | --- |
| ! (унарна) | not | !b, !(c>1 && c<9) |
| &&(бінарна) | and | a && b, a>1 && c<9 |
| || (бінарна) | or | a||b, c<0 || c>10 |

Операнди логічних операцій можуть бути логічні змінні та відношення.

Правила виконання логічних операцій надано через таблицю істинності, де true – 1, false – 0.



Операція **and** Операція **or** Операція **not**

**Логічні вирази**

Логічні (булівські) вирази складаються з  логічних операцій та операндів - логічних змінних і значень та відношень, з урахуванням пріоритетів. Для зміни порядку обчислень можна використовувати круглі дужки. Результат виконання має тип ***bool***. Правила побудови загальні - лінійна форма написання, явно вказуються знаки операцій та операнди без дублювання.

Приклади запису логічних виразів:

**x<0 || x>11**

**(i!=0 ||i>100) && (j!=i || j>0)**

**x!=0 && x!10 && x!=100**

Вираз може містити операнд, який не є логічним. Тоді значення нуль (0) сприймається як false, а значення  ≠0 - як true. Приклади:

**// логічні операції**

**bool res;**

**int a, b;**

**// операція && (AND)**

**a = 8;**

**b = 5;**

**res = a && b; // res = True**

**a = 0;**

**res = a && b; // res = False**

**// операція || (OR)**

**a = 0;**

**b = 0;**

**res = a || b; // res = False**

**b = 7;**

**res = a || b; // res = True**

**// операція ! (логічне "НІ")**

**a = 0;**

**res = !a; // res = True**

**a = 15;**

**res = !a; // res = False**

**Операції обробки окремих бітів** застосовують для обробки даних як послідовностей бітів (розрядів), кожний з яких набуває значення **0** або **1**.

& — операція бітового множення (кон’юнкція);

| — операція бітового додавання (диз’юнкція);

^ — додавання за модулем 2 (результат дії дорівнює 1, якщо число складаємих одиничних бітів непарне, якщо ж їх число парне, то результат дорівнює 0.);

~ — інвертування (побітове **NOT)**;

>> — зсув праворуч;

<< — зсув ліворуч.

**Змінна-покажчик** зберігає значення, що є адресою об’єкта в пам’яті комп’ютера. Через покажчик можна звертатися до об’єкта.

**Операції з адресами та покажчиками:**

& — одержання адреси: видає адресу змінної, ім’я якої роз­ташоване праворуч від позначення операції;

\*  — непряма адресація (розіменування): видає значення, записане за адресою, на яку посилається покажчик.

**Додаткові операції:**

**sizeof()** — знаходить розмір (у байтах) операнда, розташованого праворуч від назви операції;

**(type)** — операція приведення типу перетворює наступне за нею значення в тип, визначений ключовим словом, укладеним у круглі дужки, наприклад:

**i = i+(int)\*3.14;**

**Тернарна операція (?:)**

Тернарна операція —це операція, що має 3 операнда. Використовується як альтернатива запису умовного оператора. Синтаксис тернарної операції:

**B ? V1 : V2.**

Виконання:

1. Спочатку обчислюється значення виразу B.

2. Якщо B==true(1), то обчислюється значення виразу V1,

інакше - обчислюється значення виразу V2.

Для читабельності  рекомендується брати у круглі дужки умову та вирази. Приклади:

**(i < 1) ? 1 : i;**

**max = (a<=b) ? b : a;**

**х<0 ? -x : x ;**

**Перетворення типів**

***Правила приведення типів:***

1. Автоматично здійснюються тільки ті перетворення, які перетворюють операнди з меншим діапазоном значень в операнди з більшим діапазоном значень, наприклад:

**int i\_var = 5;**

**float f\_var = 2.5, summa;**

**…**

**summa = i\_var + f\_var;**

2. Вирази, що не мають змісту (наприклад, число з плаваючою комою в ролі індексу), не пропускаються компілятором ще на етапі трансляції:

***float f;***

***…***

***mas [f]=25; // викликає помилку трансляції (Error)***

3. Вирази, в яких могла б втрачатися інформація (наприклад, при присвоєнні довгих цілих коротшим або дійсних цілим), можуть викликати попередження (Warning), але вони допустимі:

***int i;***

***float f=3.2;***

***i=f; // попередження (Warning) при трансляції***

**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*+*º*=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.

**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 степінь, 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**(-bsqrt(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 <math. h>

|  |  |
| --- | --- |
| **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 |

**Макровизначення.**

***Макровизначення*** (*макропідстановка* або просто *макрос*)– це вираз, який при компіляції файлу з кодом програми підставляється замість символьного імені, що визначає дану макропідстановку. Макроси дозволяють замінити великий текст коротким ім'ям макросу.

Визначаються макроси директивою препроцесора ***#define***. Формат задання:

***#define ім'я\_макросу текст\_макросу***

Перед компіляцією замість імені макроса в програму підставляється його значення.

Наприклад,

#define РІ 3.14159

Дана директива вказує замінити всі появи у тексті програми символьної константи РІ на чисельну константу 3.14159. Якщо є необхідність змінити значення такої константи у всій програмі, то достатньо це зробити в одному місці програми, в директиві #define, і після повторної компіляції програми всі включення константи в програму будуть автоматично замінені.

Дозволяються також макроси з параметрами, останні підставляються в текст заміни, після чого макрос розширюється, тобто в програму підставляється текст заміни замість ідентифікатора і списку параметрів. Наприклад, для макровизначення з одним аргументом для знаходження площі круга:

#define CIRCLE\_AREA (x) ((PI)\*(x)\*(x))

Кожен раз, коли в програмному коді з’явиться CIRCLE\_AREA (x), значення х підставляється замість х в тексті заміни, символьна константа РІ автоматично заміняється її значенням (визначеним раніше), і макрос розширюється в програмі. Наприклад, оператор

area=CIRCLE\_AREA (4)

розширюється в

area=((3.14159)\*(4)\*(4)).

Макроси є спадщиною мови С, і в С++, зазвичай, замінюються більш безпечними можливостями мови, такими, як шаблони, inline-функції, перерахування enum, перевизначення типів typedef.

Символьні константи і макроси можна відмінити, використовуючи директиву препроцесора ***#undef***. Ця директива відміняє визначення символьної константи або імені макроса. Область їх дії продовжується від місця визначення в програмному коді до місця відміни визначення за допомогою директиви #undef, чи до кінця файлу, якщо дана директива відсутня. Після дії директиви #undef дозволяється повторно визначати макрос за допомогою директиви #define.

**Директиви умовної компіляції**.

Умовна компіляція дозволяє управляти виконанням директив препроцесора і компіляцією програмного коду. Директиви умовної компіляції: #if, #elif, #else та #endif. Кожна конструкція #if закінчується #endif. Умовні конструкції препроцесора, що перевіряють декілька варіантів, реалізуються за допомогою #elif (еквіваленту else if структури if) і #else (еквіваленту else структури if). Порядок використання директив умовної компіляції:

**#if константний\_вираз\_1**

**…**

**#elif константний\_вираз\_2**

**…**

**#elif константний\_вираз\_3**

**#else**

**…**

**#endif**

Кожна із умовних директив препроцесора оцінює значення цілочисельного виразу. Якщо цілий константний вираз у директиві #if має ненульове значення (TRUE), то при компіляції включаються всі наступні рядки до #elif або #endif або #else (elif діє як гілка else-if). Наприклад, змінюючи константу VERSION, можна керувати включенням файлів:

#define VERSION 3

#if VERSION == 1

#define INCLUDE\_FILE "file\_1.h"

#elif VERSION == 2

#define INCLUDE\_FILE "file\_2.h"

#else

#define INCLUDE\_FILE "file\_3.h"

#endif

#include INCLUDE\_FILE

**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 першу комірку з групи, а величину групи визначає тип змінної (рис. 1).

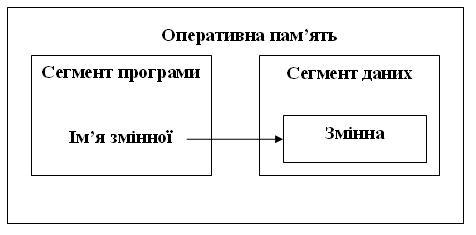


Рис. 1. Ідентифікатор змінної та його асоціація з коміркою пам'яті

**Реалізація різних алгоритмічних структур на мові С++**

При написанні програм використовують такі базові алгоритмічні конструкції :

1. Слідування – блоки алгоритму виконуються послідовно.

2. Розгалуження – в залежності від умови виконується одна або інша гілка алгоритму.

3. Цикл – група блоків алгоритму виконуються декілька разів. Обов’язковою вимогою є наявність умови виходу з циклу.

**Інструкції розгалуження**.

Інструкції розгалуження реалізують оператори вибору, до яких відносять оператор умовного переходу **if** та оператор-перемикач **switch**.

Оператор умовного переходу **if** використовується для розгалуження процесу обчислень на два напрямки і має такий два форми запису: повна та неповна.

Повна формамає вигляд

**if (*умова*) *інструкція1* else *інструкція2;***

Слова **if** та **else** є зарезервованими, дужки навколо умови обов'язкові. Цій формі оператора відповідає блок-схема на рис. 1 а). В операторі ***умова*** — це вираз, який має логічне значення (**true** — «істина» або **false** — «хиба, неправда»).

Реалізується оператор **if** в цій формі таким чином: спочатку обчислюється вираз ***умова***; якщо значення виразу не дорівнює нулю («істина»), виконується **інструкція 1**, в протилежному випадку — **інструкція 2** і далі управління передається оператору, що є наступним за умовним оператором **if**. Приклад:

**if ( i < j ) і++;**  
**else**  
**{ j = i-3; i++; }**

Неповна формамає вигляд

**if (*умова*) *інструкція1;***

В цій формі оператора **if** друга частина (тобто **else**) може бути відсутня і тоді, якщо вираз приймає значення **false** («хиба, неправда»), виконується зразу наступний оператор програми, що розташований за умовним. Таку конструкцію називають “пропуск оператора”. Цій формі оператора відповідає блок-схема на рис. 1 б). Приклад:

**if ((a>0)&&(b>0)&&(c>0)&&(a+b>c)&&(a+c>b)&&(b+c>a))**

**соut << "a,b,c –сторони трикутника"<<endl;**

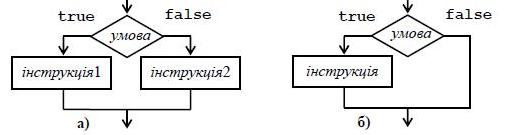


Рис.1 Блок-схеми двох форматів інструкції розгалуження

Коли у будь-якій гілці розгалуження необхідно виконати декілька операторів, їх слід розташовувати у блоці, інакше компілятор не зможе зрозуміти, де закінчується розгалуження. блок може включати різні оператори, у тому числі описи та інші умовні оператори, але не може складатися з одних описів. Потрібно ураховувати, що змінна, яка описана у такому блоці, за межами блока не існує. Синтаксис C++ припускає, що у випадку застосування вкладених умовних операторів кожне **else** відповідає найближчому до нього попередньому **if**. Наприклад:

**if (с > d** **&& (с > r** **|| с == 0)) d++;**  
**else**  
**{d** **/= с; с = 0 }**

тут декілька умов слід об’єднати знаками логічних операцій, а сукупність операцій необхідно розмістити у блоці.

Обчислення найбільшого значення серед трьох змінних **а, b, с**:

**if (а > b)  
{ if (а > с) max = а;**  
**else max = с; }**  
**else { if (b > c) max = b;**  
**else max = c; }**

У цьому фрагменті програми фігурні дужки можуть бути відсутні, тому що компілятор відносить частину **else** до найближчого **if**;

**if(p++) ps++;** — вираз не використовує операцій відношення.

**Оператор-перемикач** **switch** називають оператором множинного розгалуження. Він використовується для вибору одного а багатьох варіантів рішення і має таку форму запису:

**switch (L)**

**{**

**case к.в.1: *інструкція 1*; [break;]**

**case к.в.2: *інструкція 2*; [break;]**

**..................................................**

**case к.в.n: *інструкція* n; [break;]**

**[default: *інструкція* n+1;]**

**}**

де **switch, case, default** — службові слова;

**break** — оператор (необов’язковий) здійснює вихід з оператopa **switch**;  
**L** — будь-який вираз одного з цілих типів; цей вираз називається **селектором варіантів**.

**к.в.1, …, к.в.n** — константні вирази, які не можуть повтoрюватися і не можуть містити змінних чи викликів функцій; зазвичай, це ціла або символьна константа. Ці значеннями називаються **мітками варіантів**;

**оператор 1; …** – будь-які оператори мови C++.

У процесі виконання цього оператора спочатку обчислюється значення виразу **L**, потім це значення порівнюється (послідовно зверху донизу) зі значеннями константних виразів **к.в.1, …, к.в.n.** У випадку збігу значень **L** і одного з цих константних виразів та, якщо наприкінці гілки розгалуження (варіанту) немає оператора **break**, виконуються всі оператори, починаючи з відповідної гілки. За наявності оператора **break** виконується тільки оператор, що знаходиться у відповідній гілці розгалуження, і керування передається оператору, який розташований за межами oпeратора **switch**. Якщо значення виразу **L** не збігається з жодним із значень константних виразів, то виконується гілка **default** і здійснюється вихід з оператора **switch**. У випадку, коли в операторі **switch** відсутня гілка **default** (вона необов’язкова) і значення **L** не збігається з жодним із значень константних виразів, відбувається вихід з оператора **switch**. Варіант / гілка з міткою **default** можна записати будь-де, але рекомендується записувати його останнім.

Приклад використанням оператора **switch**:

**int а=2;**

**switch (а)**

**{**

**case 1: func1( );**

**case 2: func2( );**

**case 0: …**

**case 4: func3( );**

**default: printf ("good bye \n");**

**}**

Тут оператор **switch** передбачає реалізацію функцій **func2(), func3()** і **default: printf (“good bye \n”);.**

Фрагмент програми можна записати по-іншому:

**int а=2;**

**switch (а)**

**{**

**case 1: funcl( ); break;**

**case 2: func2( ); break;**

**case 0: ...**

**case 4: func4( ); break;**

**default: printf ("good bye \n");**

**}**

Оператор-перемикач приводить до виконання тільки гілки

**case 2: func2( ); break;** після чого здійснюється вихід із **switch**.

Щоб написати кілька інструкцій там, де за правилами мови має бути одна, наприклад, як гілку в умовній інструкції, використовують **блок** – послідовність інструкцій у дужках **{}**. Він має такий загальний вигляд:

**{**

***інструкція***

**...**

***інструкція***

**}**

Виконання блоку полягає в послідовному виконанні інструкцій, записаних у ньому.

**Прості інструкції повторення обчислень**

Циклічні, тобто повторювані, обчислення задають за допомогою оператору циклу. Який має форми:

Інструкція циклу з передумовою **while**,

Інструкція циклу з післяумовою **do while**,

Інструкція циклу **for**.

Оператори циклу використовують для здійснення багаторазового повторення деякої послідовності дій. Кожен цикл складається з тіла циклу, тобто операторів, що виконуються декілька разів. Один прохід циклу називається ітерацією.

Інструкція циклу з передумовою **while** виконується, якщо умова перевіряється до початку циклу, і має вигляд:

**while (умова) *інструкція***

Слово **while** є зарезервованим, дужки обов'язкові, **while (*умова*)** – це **заголовок циклу**, а ***інструкція*** – **тіло**.

Інструкція циклу виконується так. Спочатку обчислюється умова в заголовку. Якщо вона істинна, то виконується тіло циклу та знов обчислюється умова. Якщо вона істинна, то все повторюється. Виконання інструкції циклу закінчується, коли обчислено значення умови **false**, тобто хибність. Отже, в останньому циклі тільки обчислюється умова, а тіло не виконується. Якщо при першому обчисленні умова хибна, то тіло циклу не виконується жодного разу. **Ітерація циклу** складається з перевірки умови циклу та виконання після неї тіла циклу. Інструкції циклу з передумовою відповідає блок-схема на рис. 2.

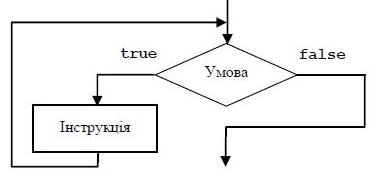


Рис. 2. Блок-схема інструкції циклу з передумовою

Умову в інструкції циклу називають **умовою продовження**, оскільки, якщо вона істинна, то виконання інструкції циклу продовжується. Цикл починається обчисленням умови, тому її ще називають **передумовою**. Інструкції циклу з передумовою застосовують зазвичай тоді, коли кількість повторень циклу наперед невідома.

Приклад: наступна послідовність операторів обчислює суму квадратів перших **n** натуральних чисел:

**int і=0, sum=0;**  
**while (і < n) sum += ++і \* і;**

Інструкція циклу з **післяумовою**, або **do**-інструкція, має загальний вигляд

**do *інструкція* while (умова);**

**Оператор циклу з післяумовою do while** звичайно застосовується у випадках, коли тіло циклу виконується хоча б один раз. Слово **do** (виконувати) є ключовим. Інструкція циклу з післяумовою виконується так: спочатку виконується тіло циклу, потім обчислюється умова. Якщо вона хибна, то цикл завершується, інакше повторюється тіло й знову обчислюється умова. На відміну від інструкції циклу з передумовою, цикл *починається діями в тілі циклу* та закінчується обчисленням умови. Умова перевіряється після виконання тіла циклу, тому її називають **післяумовою**. Тіло циклу, заданого **do**-інструкцією, виконується обов'язково хоча б один раз (на відміну від **while**-інструкції). Інструкцію циклу з післяумовою використовують, коли потрібно спочатку виконати тіло циклу, і лише потім перевіряти умову продовження. Циклу з післяумовою відповідає блок-схема на рис. 3.

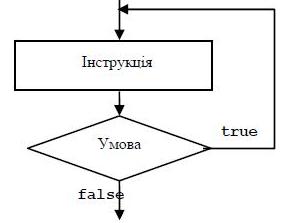


Рис. 3. Блок-схема інструкції циклу з післяумовою

Приклад.Потрібно з клавіатури ввести ціле число від 10 до 99. Якщо користувач набрав число за межами цього діапазону, то слід *повторити спробу*. Отже, спочатку треба вводити число, а потім перевіряти умову того, що число є двозначним.

**do {**

**cout << "Enter one integer in [10,99]>";**

**cin >> k;**

**} while (!(10<=k && k<=99)); // 10<=k && k<=99**

Інструкція циклу **for** або **for**-інструкція, має загальний вигляд

**for (*початкова дія*; *умова*; *перехідна дія*)**

***основна дія***

Слово **for** зарезервоване, дужки та два знаки; усередині дужок є обов'язковими. Початкова дія, умова й перехідна дія є *виразами* (кожен із них може бути порожнім), основна дія – *інструкцією*. Тілом циклу **for** називають його основну дію. Інструкція **for** виконується так само, як і інструкції вигляду

***початкова дія*;**

**while (*умова*)**

**{**

***основна дія*;**

***перехідна дія*;**

**}**

Оператор циклу **for** реалізується таким чином:

* виконується початкова дія - вираз ініціювання (виконання цієї нотації може бути здійснено до оператора **for**);
* обчислюється вираз-умова;
* якщо умовний вираз приймає значення «істина» — виконуються оператори циклу;
* обчислюється вираз ітерації;
* знову перевіряється умова;
* як тільки умова прийме значення **0** («хиба, неправда»), керування передається оператору, що розташований за оператором циклу **for**.

Оскільки в операторі **for** перевірка виразу-умови відбувається перед циклом, то у випадку помилкової умови цикл може жодного разу не виконуватися.

Оператор **for** може використовувати декілька змінних, що керують циклом, а будь-які вирази можуть бути відсутніми, наприклад:

**int n, у;**

**for (int к = 0, n = 20; к <= n; k++, n--)**

**y = k \* n;**

або

**int = і;**

**for (; і < 4; i++)**

Перший фрагмент має два вирази ініціювання і два вирази ітерації. Спочатку відбувається присвоювання значень змінним **k = 0 і** **n = 20**, далі здійснюється порівняння **k <= n** і, якщо ця умова має значення «істина», то буде виконуватися тіло циклу, а потім вираз **k++ і** **n–**, якщо ж умова не виконується, то цикл припиняє свою роботу.

C++ дозволяє поєднати ці дві дії в одному виразі – за допомогою операції послідовного обчислення. Операція зі знаком "**,**" позначає послідовне обчислення виразів, записаних через кому (в прикладі це **к = 0, n = 20;)**. Ця послідовність виразів розглядається як один вираз; його значенням є значення останнього виразу. Операція послідовного обчислення дозволяє на місці одного виразу записати кілька.

Операторам циклів с параметром **for** потрібно віддати перевагу при організації циклів з лічильниками.

В циклі **for** можна використовувати інструкції **break** та **continue.** Інструкція **break** у тілі циклу **for** завершує його виконання, а інструкція **continue** завершує виконання лише тіла циклу; відразу після неї виконується перехідна дія.

**Приклад.** Дуже часто інструкція циклу **for** зустрічається у вигляді

**for (k=0; k<n; ++k) *інструкція***

й задає виконання *інструкції* за значень *k* = 0, 1, 2, …, *n*-1 або у вигляді

**for (k=1; k<=n; ++k) *інструкція***

й задає виконання *інструкції* за значень *k* = 1, 2, …, *n*, або у вигляді

**for (k=n; k>0; --k) *інструкція***

й задає виконання *інструкції* за значень *k* = *n*, *n*-1, …, 2, 1. Змінну **k** у цих ситуаціях інколи називають **лічильником циклу**.

**Переривання break та продовження циклу continue**

Виконання інструкції **break** всередині циклу будь-якого різновиду перериває й завершує цикл; далі виконуються дії, наступні за цим циклом. Якщо **break** записано в інструкції циклу, вкладеній в іншу інструкцію циклу, то виконання **break** завершує вкладений цикл, а зовнішній цикл продовжується.

Інструкція **continue** всередині циклу задає перехід на кінець тіла циклу. В інструкціях циклу з перед- і післяумовою після **continue** обчислюється умова продовження циклу.

Приклад.За допомогою клавіатури вводиться послідовність дійсних чисел. Потрібно підрахувати суму її додатних елементів, а за появи 0 видати накопичену суму й завершити роботу.

Запрограмуємо цикл, в якому вводиться й обробляється послідовність чисел. Уведене число зберігаємо в змінній **x**, а суму додатних елементів – у змінній **sum**. Якщо під час уведення трапилася помилка, то подальші дії з уведення не виконуються, а змінна **x** зберігає своє останнє значення. Тому умовою продовження циклу буде саме відсутність помилок (інакше можна отримати цикл, який ніколи не завершиться!). Цю умову задає значення виразу введення **cin>>x**, перетворене до логічного типу.

**#include <iostream>**

**using namespace std;**

**int main()**

**{ double x;**

**double sum=0;**

**cout<<"Enter reals:\n";**

**while (cin>>x){**

**if (x==0.) break; //виходимо з циклу**

**if (x<0.) continue; //пропускаємо від'ємні**

**sum+=x;**

**}**

**cout << "sum=" << sum << endl;**

**system("pause");**

**return 0;**

**}**

Використання інструкції **continue** в цій програмі є дуже штучним. Ще одним недоліком є те, що в кінці не повідомляється, чи були помилки під час уведення. Інструкції програми виконуються *в порядку їх запису в програмі*. Про таку програму кажуть, що вона **структурована**. Інструкції **break** і **continue** *порушують* цей порядок обчислень, заплутуючи текст програми. Тому, користуючися ними, програміст повинен ретельно відслідковувати точку програми, якою продовжуються обчислення. Інколи ці інструкції дійсно скорочують запис розгалужень у циклі, проте в більшості випадків ті ж самі дії *можна описати без них*. Тому краще не зловживати **break** і **continue**.

**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)º**--**. 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.

***Контрольні запитання для самоперевірки***.

1. Що включає в себе алфавіт мови С++?
2. Що таке лексеми, як вони складаються, що до них відносять?
3. Дайте визначення ідентифікатору, яким чином він утворюється?
4. Що називають ключовими словами?
5. Які символи можуть бути роздільниками лексем?
6. Які коментарі використовує мова C++? Надайте поради до їх створення, наведіть приклади.
7. Що являє собою структура програми на C++? Наведіть основні вимоги, які слід ураховувати при створенні програм мовою C++.
8. Які основні та складені типи даних має мова С++?
9. Що таке змінна і як здійснюється її опис та визначення?
10. Що таке «область дії ідентифікатора» та «клас пам’яті»?
11. Які константи налічує С++? Охарактеризуйте їх застосування.
12. Що таке пріоритет операції? Наведіть приклади арифметичних та логічних операцій.
13. Які операції присвоювання та операції відношення налічує С++?
14. Що реалізують логічні операції та операції обробки окремих бітів?
15. Які операції над покажчиками та додаткові операції має С++?
16. Визначте основні частини типової структури програми на С++.
17. Що таке макровизначення? Наведіть приклади.
18. Для чого використовується умовна компіляція?
19. Що може міститися в зовнішніх оголошеннях?
20. Надайте визначення функції.
21. Які значення може містити умова, що перевіряється?
22. Як визначається результат операція **not?**
23. Як визначається результат операція **and?**
24. Як визначається результат операція **or?**
25. Поясніть, що представляє собою логічний вираз.
26. Як визначається тернарна операція?

**Для самостійного вивчення** *(4 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

**Рекомендована література**

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL: <http://www.ph4s.ru/bookprogramir_1.html>
3. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL: <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>