РОЗДІЛ 2
ЕЛЕМЕНТИ МОВИ С++

2.1. З історії створення мови С++

Історія мови С++ (Сі-плюс-плюс) почалася з мови С (Сі). Компанія Bell Laboratories на початку 1970-х рр. створила мову С як інструмент розробки операційної системи UNIX. Спочатку Мартін Річардс розробив мову BCPL, потім на її основі Кен Томпсон – мову B, і нарешті, Денніс Річі – мову С. Завдяки лаконічності, виразній потужності й надійним компіляторам ця мова програмування дуже швидко стала однією з найпопулярніших і найпоширеніших. Проте з часом розміри й вимоги до програм зростали, тому можливості мови С та інших мов, подібних до неї, поступово вичерпувалися.

Кризу в програмуванні, що намітилася в кінці 1970-х рр., було подолано за допомогою, головним чином, *об'єктно-орієнтованого програмування*. Однією з об'єктно-орієнтованих мов була мова С++, яку на базі мови С розробив Бйорн Страуструп у компанії Bell Laboratories у кінці 1970-х рр.

У наступні два десятиліття мову С++ було збагачено й стандартизовано. Її втілено в кількох сучасних системах програмування, зокрема в Microsoft Visual C++ (існує кілька версій цього дуже потужного засобу програмування), і на неї орієнтуються автори в прикладах цієї книги.[3]

[3] Є також інші системи програмування на базі С++, наприклад Dev-C++ або Code::Blocks, які мають відмінності в інтерфейсі й синтаксичних деталях. Зауважимо: Microsoft Visual C++ має найкращу довідкову систему.

Нащадками мови C++ є мови Java (Джава) та C# (Сі-шарп або Сі-діез), спеціалізовані для програмування в сучасних комп'ютерних мережах. Ці мови за структурою схожі на C++, тому, володіючи C++, неважко перейти на Java або C#, і навпаки.

2.2. Перша програма – "Hello, World!"

Програма мовою C++ записується у файл з розширенням **.cpp**, наприклад **prog001.cpp**.

Приклад. У книжках із мов програмування дуже часто першою наводять програму, яка "вітає світ", тобто виводить на екран рядок-привітання.

```
#include <iostream>
using namespace std;
int main ()
{
   cout<<"Hello, World!";
   return 0;
}
```

prog001.cpp

У першому рядку записано **директиву препроцесора**. **Препроцесор** – це складова частина компілятора, яка проводить попередню обробку програми. Директиви записують в окремих рядках і починають символом #. Слово **include** (включити) означає, що препроцесор перед компіляцією програми має включити в неї вміст спеціального файлу зі складу системи програмування, ім'я якого **iostream** записане в кутових дужках. У цьому файлі оголошено засоби введення й виведення (ім'я **cout**, операцію **<<** і багато інших). Без включення цього файлу ім'я **cout** буде невизначеним, і 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. У директивах імена стандартних заголовних файлів записуються в кутових дужках. Багато стандартних заголовних файлів має порожнє розширення, для решти традиційно використовують розширення **h**.

У другому рядку розташовано інструкцію компілятору "використати простір імен **std**". Не пояснюючи значення слів "простір імен" (детальніше див. підрозд. 9.3), скажемо лише, що простір імен **std** є стандартним. У сучасних системах програмування мовою C++ у ньому описано всі бібліотечні засоби останнього покоління. Завдяки наведеній інструкції спрощується доступ до бібліотечних засобів (один з них, з ім'ям **cout**, використовується в програмі). Проте компілятори попереднього покоління цієї інструкції "не розуміють", тому для них писати її не можна.

Наведена програма складається з однієї функції з ім'ям **main**. Слова **int main()** у третьому рядку – це **заголовок функції**. Дужки **()** після імені **main** указують, що це ім'я саме функції. Ім'я **int** перед іменем функції є скороченням слова integer і означає, що функція має повертати ціле значення.

Функція з ім'ям **main** називається **головною функцією**. Вона має бути в кожній програмі, адже саме з неї починається виконання програми й зазвичай нею закінчується. Ім'я **main** не є зарезервованим, але використовувати його з іншим призначенням не слід.

Вміст рядків 4–7 утворює **тіло** функції, що починається символом **{** і закінчується **}**. У тілі функції задано дії у вигляді **послідовності інструкцій**. Інструкція в п'ятому рядку задає виведення на екран повідомлення **Hello, World!**. Воно з'являється у вікні програми, яке має відкритися на екрані під час її виконання та зникнути після завершення. Текстове повідомлення, що виводиться на екран, записується в лапках **""**.

За інструкцією в шостому рядку функція має повернути значення 0 (нуль)[4].

Отже, запустимо програму на виконання. Чи встигнемо ми щось побачити? Відповідь залежить від системи програмування, що використовується. Деякі, але не всі, системи програмування дозволяють переглядати вікно програми після її завершення. З

[4] Повернути можна також інше ціле значення. Значення, що повертається функцією **main**, передається операційній системі як результат виконаної програми. За традицією 0 є ознакою вдалого виконання програми.

іншого боку, виконання програми можна затримати за допомогою бібліотечної функції **system.**

```
#include <iostream>
using namespace std;
int main () {
   cout<<"Hello, World!";
   cout<<endl;
   system("pause");
   return 0;
}
```

prog002.cpp

За інструкцією `system("pause");` виконання програми призупиняється й на екрані з'являється повідомлення, що треба натиснути будь-яку клавішу. Після натискання програма завершується. Завдяки попередній інструкції `cout<<endl;` повідомлення виводиться в новому рядку. Якби цієї інструкції не було, повідомлення з'являлося б відразу після слів `Hello, World!`. ◀

Наведена програма складається з двох директив і головної функції. Узагалі програма може містити багато функцій, директив і деяких інших елементів.

Вправа 2.1. З'ясуйте, що буде виведено за нижченаведеними програмами. Поясніть різницю між ними.

```
#include <iostream>
using namespace std;
int main() {
 cout<<"Microsoft";
 cout<<"Visual";
 cout<<"Studio";
 cout<<endl;
 system("pause");
 return 0;
}
```

```
#include <iostream>
using namespace std;
int main() {
 cout<<"Microsoft ";
 cout<<"Visual ";
 cout<<"Studio";
 cout<<endl;
 system("pause");
 return 0;
}
```

```
#include <iostream>
using namespace std;
int main() {
 cout<<"Microsoft";
 cout<<endl;
 cout<<"Visual";
 cout<<endl;
 cout<<"Studio";
 cout<<endl;
 system("pause");
 return 0;
}
```

2.3. Алфавіт і словник мови С++

Кожна мова має **алфавіт** – скінченну множину символів, дозволених для використання. Алфавіт для написання програм мовою С++ містить:

– **літери** (**букви**) – великі й малі латинські **A, B, … Z, a, b, … z,** а також символ підкреслення _ та "долар" **$**;

– **десяткові цифри** **0, 1, 2, …, 9;**

– **знаки пунктуації** **+ - * ~ / \ % ? ! = < > & | ^ # . , ; : ' " ( ) [ ] { }** ;

– деякі інші символи.

Із символів алфавіту утворюються **лексеми** – "слова", тобто послідовності символів, що розглядаються як неподільні й мають власний зміст. Серед лексем мови С++ виділяють константи, імена, знаки операцій тощо.

Константа (*constant*) – це позначення значення (числового або іншого). Константи також називають **літералами** (*literals*).

Нам добре відомі числові константи, наприклад **273, 3.1415926, 2.71828** (ціла частина від дробової відокремлюється крапкою, а не комою). В апострофах записуються символьні константи, у лапках – рядкові, наприклад **'3'**, **'A'**, **'\n'**, **"University"**, **"3.1415926"**, **""**. Докладніше константи описано нижче.

Ім'я – це послідовність літер алфавіту й цифр, що починається з літери[5], наприклад **A, x1, _1a$2, best_student.**

Великі й малі літери в іменах *відрізняються*: **Name, NAME, name** – це різні імена. Довжину імен *не обмежено*[6].

Ім'я завжди *іменує* якийсь об'єкт, видялючи з-поміж інших, тобто *ідентифікує* його, тому ім'я ще називають **ідентифікатором**.

[5] Символ підкреслення _ може бути першою літерою імені, але імена, що починаються з одного чи двох _, мають у С++ спеціальне призначення. Тому використовувати імена з _ на початку не рекомендується.

[6] Формально довжину імен у С++ не обмежено, але, наприклад, компілятор Microsoft Visual C++ 2005 розрізняє лише перші 2048 символів імені.

Деякі імена (англійські слова або їх скорочення) мають спеціальний зміст, наприклад **`const`**, **`else`**, **`int`**, **`sizeof`**. Вони називаються **зарезервованими словами**. Використовувати їх за іншим призначенням не можна (цього просто не дозволить компілятор). Зарезервовані слова інколи вважають окремим різновидом лексем.

Знак операції, або **оператор** (*operator*) – це позначення операції, виконання якої над числами та іншими значеннями породжує число або інше значення. Значення, до яких застосовується операція, називаються її **операндами** (*operand*), а породжуване значення – **результатом** (*result*).

Операції позначають окремими символами, наприклад **`+`** або **`-`**, а також іменами або послідовностями інших символів, наприклад **`sizeof`** або **`<=`**.

Лексеми часто відокремлюються так званими **порожніми символами**, які не мають зображення. Вони з'являються в тексті програми, якщо, набираючи його, натискати клавіші **`Space`** (пробіл), **`Enter`** (кінець рядка) і **`Tab`** (символ табуляції). Будь-які дві лексеми можна відокремити порожніми символами в довільній кількості. Компілятор, обробляючи текст програми з деякої поточної позиції, завжди виділяє найдовшу можливу лексему. Тому порожні символи між сусідніми лексемами обов'язкові лише тоді, коли, записані разом, ці лексеми утворюють лексему. Наприклад, записи **`1+2`**, **`sizeof-2`** та **`1<=2`** позначають застосування відповідних операцій і не є лексемами, а **`sizeof2`** є ім'ям, тобто **однією** лексемою.

2.4. Поняття типу даних

Комп'ютер може зображувати та обробляти дані різної природи. Наприклад, рік народження людини зображують цілим числом, вага може бути не лише цілою, але й дробовою, а прізвище позначають послідовністю символів. Числа можна додавати й віднімати, послідовності символів – дописувати одну до одної. Отже, різні за природою дані (значення) мають різні спо-

соби зображення й допустимі операції, і за цими ознаками поділяються на типи.

2.4.1. Типи даних

Тип даних – це множина значень разом із множиною застосовних до них операцій. Множина значень називається **носієм** типу, множина операцій – **сигнатурою**.

У програмуванні використовуються типи чисел, символьних і логічних значень. Ці значення розглядаються як цілісні елементи, що не мають окремих складових частин, тому називаються **скалярними** на відміну від **структурних**, складених з окремих компонентів. Типи скалярних даних називаються **скалярними**. Кожна мова програмування забезпечує цілу сім'ю скалярних типів. Вони мають певні, означені наперед (стандартні) імена й називаються **базовими скалярними типами** мови.

Серед базових скалярних типів мови C++ є типи цілих і дійсних чисел з іменами **`int`** і **`double`**, а також тип символів **`char`** (*int* і *char* – скорочення від *integer* і *character*, *double* – подвійний) і логічний тип **`bool`** (*bool* – скорочення від *boolean*). Крім того, часто користуються також типом беззнакових цілих чисел **`unsigned int`**, або просто **`unsigned`**.

2.4.2. Мінімальні відомості про базові типи

Типи **`int`**, **`unsigned`**, **`char`** та **`bool`** у мові C++ відносять до **цілих** (*integral*) типів. Числа типу **`int`** зображуються в знаковій формі й займають 4 байти, тому носій цього типу утворено цілими числами від -2147483648 до 2147483647 (див. с. 14). Числа типу **`unsigned`** (перекладається як *беззнаковий*) мають зображення без знака, тому їх діапазон – від 0 до 4294967295. Мінімальне й максимальне значення типу **`int`** позначено іменами **`INT_MIN`** та **`INT_MAX`**, максимальне значення типу **`unsigned`** – **`UINT_MAX`**.

Значення символьного типу **`char`** займають 1 байт; усього їх 256. Логічний тип **`bool`** має два значення, що позначаються іменами **`false`** і **`true`** (хибність та істина) і зображуються в ком-

п'ютері числами `0` та `1`, відповідно. Значення "хибність" та "істина" називаються **булевими** на честь видатного англійського логіка й математика Джорджа Буля.

Дійсні числа типу `double` займають 8 байтів, тому їх множина містить майже 2^{64} чисел (кілька з усіх можливих комбінацій нулів і одиниць не є зображеннями чисел). Ця *обмежена скінченна* множина чисел є симетричною відносно числа 0. Найбільше й найменше відмінні від 0 додатні числа позначено іменами `DBL_MAX` та `DBL_MIN`. Ці числа дорівнюють приблизно 10^{308} та 10^{-308}. Зауважимо: усі цілі числа типу `int` мають зображення в типі `double`.

Цілі та дійсні типи разом називають **арифметичними.**

2.5. Різновиди констант

У цьому підрозділі наведено мінімальні відомості про вигляд констант мови C++ і засоби їх виведення.

2.5.1. Символьні константи

Символьна константа – це символ в апострофах (`'A'`, `'1'`, `'.'`) або два символи в апострофах, першим з яких є \ (зворотна скісна, або *backslash*). Наприклад, константи `'\''`, `'\"'`, `'\\'` позначають символи, що називаються "апостроф", "лапки" та "зворотна скісна". Є кілька констант, наприклад `'\n'` або `'\t'`, що містять малу латинську літеру (**керувальні символи**). Вони використовуються в спеціальний спосіб під час виведення на зовнішні носії даних. Зокрема, `'\n'` задає перехід на новий рядок екрана.

Символьні константи зображують значення типу `char`, які займають один байт. Цей байт як зображення числа без знака дає **код символу** (число від 0 до 255), а зі знаком – від -128 до 127. Відповідність між числами від 0 до 127 і символами зафіксовано в *Американському стандартному коді для обміну інформацією* (так звана *таблиця ASCII*), решті кодів можуть відповідати різні

набори символів. Наприклад, константа `' '` (пробіл) має код 32, а `'\0'` – код 0. Окремими властивостями таблиці ASCII є такі:

– символам `'0'`, `'1'`, ..., `'9'` відповідають послідовні коди від 48 до 57;

– символам `'A'`, `'B'`, ..., `'Z'` – від 65 до 90;

– символам `'a'`, `'b'`, ..., `'z'` – від 97 до 122.

Інструкція `cout<<'Z';` виводить значення константи `'Z'` – на екрані з'являється `Z`. Аналогічно можна вивести інші символьні константи. Для того, щоб наступне повідомлення виводилося з нового рядка, можна скористатися інструкцією `cout<<endl;` або `cout<<'\n';` .

За інструкцією вигляду `cout<<`*`константа`*`;` обробляється не константа, а зображене нею значення. Послідовність символів, утворена за значенням, може відрізнятися від константи.

Вправи

2.2. Що буде виведено за програмою?

```
#include <iostream>
using namespace std;
int main() {
   cout<<'1';  cout<<'\n';
   cout<<'Y';  cout<<'e';
   cout<<'s';  cout<<endl;
   system("pause"); return 0;
}
```

2.3. Написати програму, що друкує символи `'`, `"`, `\`.

2.5.2. Рядкові константи

Рядкові константи позначають послідовності символів і записуються в лапках, наприклад `"You are welcome!"`. Символи `'`, `\`, `"` у рядковій константі записують як `\'`, `\\`, `\"`, відповідно. До значення, заданого рядковою константою, у пам'яті додається символ `'\0'`, який позначає кінець послідовності символів. Окрім того, якщо в константі присутній символ `'\0'`, то значенням, яке вона задає, є послідовність символів перед `'\0'`. На-

приклад, константі `"\"A'BC\"\0zz"` відповідає послідовність байтів із символами

```
"  A  '  B  C  "  '\0'  z  z  '\0'
```

Значення утворене першими шістьма символами `"A'BC".` Саме воно виводиться за інструкцією `cout<<"\"A'BC\"\0zz";` .

За допомогою рядкових констант перепишемо програму зі вправи 2.2.

```
#include <iostream>
using namespace std;
int main() {
   cout<<"1\nYes\n";
   system("pause"); return 0;
}
```

prog003.cpp

Вправи

2.4. Написати програму, що виводить ім'я дискового каталогу, в якому встановлено систему програмування C++.

2.5. Поясніть різницю між значеннями `"a"` та `'a'`.

2.5.3. Цілі константи

Цілі константи позначають цілі числа й мають десяткову, вісімкову й шістнадцяткову форму запису (про системи числення див. додаток А). У мові C++ цілі константи невід'ємні. **Десяткова константа** – це, як і в математиці, послідовність десяткових цифр, що не починається з `0`: `273`, `1024` тощо. Запис **шістнадцяткових констант** починається символами `0x` або `0X`. Наприклад, константа `0x11` позначає число 17, а `0xf` та `0xF` – число 15. Ціла константа, що починається з `0` й далі містить цифри від 0 до 7, є **вісімковою**. Наприклад, константи `0` та `010` позначають числа 0 та 8, відповідно.

Якщо в програмі спеціально не вказано інше, значення цілих констант незалежно від форми їх запису *виводяться в десятковому записі*. Наприклад, за інструкцією `cout<<11;` на екран виводиться `11`, а за `cout<<0x11; – 17`.

Вправи

2.6. Пояснити різницю між а) `0`, `'\0'` та `'0'`; б) `123` та `"123"`.

2.7. Що виводиться на екран за такими двома програмами? Поясніть різницю між ними.

```
#include <iostream>
using namespace std;
int main() {
 cout<<13; cout<<'\n';
 cout<<013;  cout<<'\n';
 cout<<0x13; cout<<endl;
 system("pause");
 return 0;
}
```

```
#include <iostream>
using namespace std;
int main() {
 cout<<"13"; cout<<'\n';
 cout<<"013"; cout<<'\n';
 cout<<"0x13"; cout<<endl;
 system("pause");
 return 0;
}
```

2.5.4. Дійсні константи

Розглянемо приклад. Число 12,34 можна позначити, наприклад, так: $123{,}4 \cdot 10^{-1}$. У цьому записі є ціла частина `123`, дробова частина `,4` і десятковий порядок `-1`. Запису відповідає константа `123.4E-1`, в якій `123` – *ціла* частина, `.4` – *дробова*, а `E-1` – *порядок*. Це саме число можна зобразити й іншими константами: `12.34`, `1234.e-03`, `1234e-03`, `.1234E2`, `0.01234E+3` тощо. Отже, ціла частина – це непорожня послідовність цифр, дробова – послідовність цифр із *крапкою* на початку, порядок – латинська літера `E` або `e` з однією або кількома цифрами, можливо, зі знаком `+` або `-`.

Константа, в якій ціла частина містить одну цифру від `1` до `9`, називається **нормалізованою**, наприклад `1.234E+1`, `9.81` або `1.0E2` (для числа 0 такою є `0.0`).

Дійсні константи записуються тільки як десяткові. Якщо константа має цілу частину, то за нею йде дробова частина й порядок (можливо, одне з них). При цьому в дробовій частині, якщо вона є, цифр може не бути, але крапка обов'язкова. Якщо цілої частини немає, то константа починається крапкою, після якої має бути хоча б одна цифра. Порядок необов'язковий. Приклади наведено вище. Перед константою може бути знак `-`; тоді вона задає від'ємне число: `-1.2345E1`.

Значення дійсних констант виводяться в різних формах (*форматах*). Значення, зображені дійсними константами, виводяться або з **фіксованою** (*fixed*) **крапкою**, або в **нормалізованій**

(**науковій** – *scientific*) **формі**; наприклад, для числа 12,34 це відповідно `12.34` та `1.234e+001`. Від чого залежить формат виведення та як ним керувати, див у п. 3.3.2.

Вправи

2.8. Поясніть, у чому різниця між `314.` та `314`.

2.9. Поясніть, у чому різниця між `12.151` та `"12.151"`.

2.6. Поняття змінної величини

2.6.1. Змінні величини

Поняття змінної числової величини з'явилося в роботах Декарта в XVII ст. Пізніше з'ясувалося, що змінна величина може бути не лише числовою. У найширшому розумінні **змінна величина** – це загальне поняття реального чи уявного об'єкта (або його окремої характеристики), який може перебувати в різних станах. Змінні величини позначають *іменами*; наприклад, у другому законі Ньютона $a = F/m$ імена m, a, F позначають змінні – масу тіла, прискорення його руху й силу, що діє на нього. У програмуванні змінна є моделлю (зображенням) об'єкта в пам'яті комп'ютера.

Змінна в машинній програмі – це ділянка пам'яті, яка може мати різні стани й зображувати ними різні стани об'єкта.

Змінну в програмі мовою високого рівня позначає **ім'я**. Кожна ділянка пам'яті комп'ютера має **адресу** (номер першого байта ділянки), і цією адресою позначається в машинній програмі. Кажуть, що адреса *вказує*, або *посилається* на ділянку. Під час трансляції програми ім'я змінної перетворюється на адресу деякої ділянки пам'яті. Отже, вважатимемо, що під час виконання програми ім'я змінної *вказує* на ділянку пам'яті (рис. 2.1).

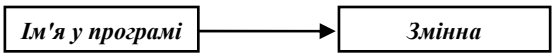


Рис. 2.1. Змінна та її ім'я

Станам об'єкта, зображуваного змінною, відповідають її **значення** (числові та інші), що належать певному типу. Тип

змінної та її ім'я задаються в **інструкції оголошення імені змінної**, що має вигляд

```
ім'я-типу  ім'я-змінної;
```

Після імені типу можна записати кілька імен змінних через кому. Інструкція оголошення імені змінної одночасно є **означенням змінної**, тобто задає ділянку пам'яті, на яку вказує ім'я змінної.

Приклади.

1. Інструкції

```
int numberOfStudents;
double radius, circleLength;
```

задають утворення ділянки пам'яті, яка ідентифікується іменем `numberOfStudents` і може зберігати цілі числа, і двох ділянок з іменами `radius` і `circleLength`; їх значеннями можуть бути дійсні числа.

2. Квадратне рівняння $ax^2+bx+c=0$ визначається трьома дійсними коефіцієнтами a, b, c. Щоб зобразити рівняння, означимо три змінні: `double a, b, c;`. ◂

Імена змінних та інших об'єктів обирають так, щоб вони були пов'язані з об'єктами або характеристиками, які вони зображують, наприклад `temperature`, `radius`, `circleLength` (температура, радіус, довжина кола).

Завдяки змістовним іменам текст програми стає зрозумілішим.

Ім'ям змінної не може бути зарезервоване слово. Імена, означені в бібліотеках системи програмування, використовувати як імена змінних можна, але краще цього не робити.

Надалі будемо дотримуватися таких угод. Ім'я змінної починаємо з малої літери; якщо воно утворене з кількох слів, то всі слова, окрім першого, записуємо з великої літери, наприклад `circleLentgh`. Ця система запису нині є найпоширенішою у світі й через зовнішній вигляд імен має назву **"верблюжий запис"** (*camel notation*).

Якщо потрібна невелика й відома наперед кількість змінних, що позначають однотипні сутності, то в кінці імені можна дописати номер, наприклад `radius1`, `radius2`.

Існує давня традиція, за якою імена `i`, `j`, `k`, `m`, `n` дають змінним цілих типів. Найчастіше ці змінні позначають індекси (номери) або кількості.

Узагалі, змінна може бути складовою частиною іншої змінної. Тоді вона позначається не ім'ям, а деяким складнішим виразом. Однак поки що не будемо на цьому зосереджуватися.

Вправи

2.10. Написати означення змінних, що зображують: а) день, місяць і рік народження; б) точку площини; в) клітинку шахового поля; г) дріб; д) квадратне рівняння; е) прямокутник на площині.

2.11. Пояснити різницю між: а) `a` та `'a'`; б) `a` та `"a"`.

2.6.2. Уведення значень у змінні й виведення їх на екран

Є два способи надати змінній значення – увести із зовнішнього носія або присвоїти. У найпростішому випадку **інструкція введення** значення в змінну має вигляд

```
cin >> ім'я-змінної;
```

і задає введення з клавіатури. Ім'я `cin`, як і `cout`, оголошене у файлі `iostream`. Виконуючи операцію введення, комп'ютер зупиняється й очікує на значення для змінної. У відповідь слід на клавіатурі набрати деяку послідовність символів, що зображує значення (ці символи з'являться на екрані), і натиснути на клавішу `Enter`. Після цього за введеною послідовністю (вхідною константою) створюється відповідне значення та присвоюється змінній. За стандартних налаштувань запис числа сприймається як десятковий.

Якщо натиснути `Enter`, не набравши нічого, окрім пробілів, то комп'ютер і надалі чекатиме. Узагалі, перед вхідною константою можна набрати довільну кількість порожніх символів, що відповідають клавішам пробілу, табуляції та `Enter`. За стандартних налаштувань усі порожні символи буде пропущено.

В інструкції введення можна записати кілька імен змінних, кожне після відповідного знака `>>`. За виконання такої інструкції

треба набрати на клавіатурі відповідну кількість вхідних констант, відокремивши їх одним або кількома порожніми символами.

Приклади

1. Нехай є змінна `char c;` і виконується інструкція `cin >> c;`. Щоб присвоїти змінній значення `'A'`, треба натиснути послідовно на клавіші **A** та **Enter**, а значення `'7'` – на клавіші **7** та **Enter**. Якщо перед клавішею **A** або **7** кілька разів натиснути на клавіші **Enter** або **Space**, то результат буде той самий.

2. Нехай змінні `double a, b, c;` зображують коефіцієнти квадратного рівняння. Увести в них дійсні значення можна за допомогою трьох окремих інструкцій:

```
cin >> a; cin >> b; cin >> c;
```

Ті самі дії буде задано й однією інструкцією:

```
cin >> a >> b >> c;
```

В обох ситуаціях на клавіатурі слід набрати три константи, натискаючи між ними на клавіші пробілу, табуляції або **Enter**. Виконання інструкцій закінчиться, коли після третьої константи буде натиснуто на **Enter**. ◂

Практично перед кожною інструкцією введення з клавіатури варто записати інструкцію виведення, яка запрошує до введення значень і вказує, скільки та яких типів.

Приклад. Уведення значень у дійсні змінні, що зображують коефіцієнти квадратного рівняння, можна оформити таким чином:

```
cout << "Enter coefficients of quadratic equation ";
cout << "(three real or integer numbers):\n";
cin >> a >> b >> c;
```

Після появи цього запрошення курсор буде переведено в наступний рядок екрана, і з нього почнеться відображення символів, набраних на клавіатурі. ◂

Вивести значення змінної на екран можна за допомогою інструкції вигляду

```
cout << ім'я-змінної;
```

Приклад. Після введення значень трьох дійсних змінних за інструкцією

```
cin >> a >> b >> c;
```

можна вивести їх на екран, відокремивши пробілами:

```
cout << a; cout << ' '; cout << b;
cout << ' '; cout << c;
```

Ті самі дії можна задати й однією інструкцією:

```
cout << a << ' ' << b << ' ' << c;
```

Якщо введено 1, –3, 2, то буде надруковано `1 -3 2.` ◂

Вправа 2.12. Що буде виведено на екран, якщо під час виконання програми введено символи `15 n` та натиснуто на клавішу `Enter`?

```
#include <iostream>
using namespace std;
int main() {
   int n;  char a;
   cin >> n >> a;
   cout<<"n="<<n<<' '<<"a="<<a<<endl;
   system("pause"); return 0;
}
```

2.6.3. Присвоювання: операція, інструкція, вираз

Надати значення змінній можна за допомогою **операції присвоювання значення змінній**, яка копіює значення деякого виразу в змінну. Знаком операції є `=`, її можна задати в **інструкції присвоювання** (*assignment statement*):

```
ім'я-змінної = вираз;
```

Спочатку обчислюється вираз праворуч, потім його значення записується в змінну, указану ліворуч. Найпростішими виразами є константи та імена змінних, складніші описано нижче.

Приклади

1. Якщо є символьна змінна `char c;`, то інструкція `c='0';` надає їй значення `'0'`.

2. Якщо є ціла змінна `int i;`, то перша з інструкцій `i=13;` `cout<<i;` надає їй значення 13, а друга виводить символи `13` на екран.

3. Нехай дійсні змінні `double a, b, c;` зображують квадратне рівняння $ax^2+bx+c=0$. Щоб задати рівняння $x^2-3x+2=0$, присвоїмо їм значення відповідно 1, -3, 2, а потім виведемо на екран.

```
a=1.0; b=-3.0; c=2.0;
cout << a << ' ' << b << ' ' << c;
```

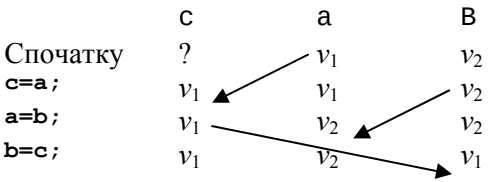
Під час виконання має бути надруковано `1 -3 2.` ◂

В інструкції присвоювання ***ім'я-змінної = вираз;*** ліворуч ім'я змінної позначає ділянку пам'яті, в яку значення записується, а у виразі праворуч – значення, що добувається з ділянки. Добування значення змінної називається її **розіменуванням**.

Приклади

1. Якщо змінні `a` та `b` однотипні, то інструкція `a = b;` копіює значення змінної `b` у змінну `a`.

2. Щоб обміняти місцями значення двох однотипних змінних `a` та `b`, скористаємося третьою змінною `c` того самого типу. Припустимо, що змінні `a` та `b` мають початкові значення v_1 і v_2, і праворуч укажемо значення, яких набувають змінні після виконання інструкцій. Початкове значення змінної `c` вважається невизначеним і позначене знаком `?`.



Перед тим, як використовувати значення змінної, *необхідно забезпечити*, щоб раніше ця змінна його отримала (під час присвоювання або введення), інакше можливі непередбачувані наслідки. Зазвичай компілятори лише попереджають, що змінна не отримала значення перед використанням, тому за цим має стежити програміст.

Вправа 2.13. Написати інструкції оголошення дійсних змінних `a`, `b`, `c`, присвоювання їм початкових значень та обміну місцями цих значень за колом: значення `a` пересувається до `c`, `c` до `b`, `b` – до `a`. *Порада*: почніть із проектування коду.

Запис вигляду *ім'я-змінної = вираз* (без ; в кінці) називається **виразом присвоювання**. Його значенням є значення, присвоєне змінній.

Вираз присвоювання можна записати праворуч від знака `=`, наприклад, `a = b = 2`. Для наочності "внутрішнє" присвоювання можна записати в дужках: `a = (b = 2)` – дії будуть ті самі. Інструкція `a = b = 2;` задає в програмі ті самі дії, що й

послідовність `b = 2; a = b;`. Довжину ланцюгів присвоювань не обмежено.

Інструкції та вирази присвоювання можуть бути частиною інструкцій інших різновидів (наприклад, інструкцій керування), описаних нижче.

Сукупність змінних, імена яких означено в програмі, називається **пам'яттю програми**[7], а відповідність між іменами змінних та їх станами – **станом пам'яті програми**. Присвоювання значення змінній **змінює стан пам'яті програми**.

Ім'я змінної є прикладом загальнішого поняття – ***l*-вираз**. Так називають вираз, який можна записати ліворуч (*left*) у виразі присвоювання. Він позначає змінну. Вираз, що позначає значення й записується праворуч (*right*) у виразах присвоювання, називається ***r*-виразом**.

2.6.4. Ініціалізація змінних

Присвоїти початкове значення змінній можна в її означенні (**ініціалізувати змінну**), дописавши до імені змінної знак `=` і вираз. У виразі можуть бути константи та імена змінних (звичайно, якщо вони вже отримали значення).

Приклади

1. За інструкцією `int a = 44, b = a, c;` створені змінні `a` та `b` отримують значення `44`, а значення змінної `c` залишається невизначеним, тому використовувати її значення можна тільки після того, як вона його отримає.

2. Інструкція `int i=j=k=0;` створює змінні `i`, `j`, `k`, які відразу отримують значення `0`. ◂

Вправа 2.14. Коло на площині можна задати його центром і радіусом. Написати програму, в якій змінні, що зображують коло на площині, оголошені та ініціалізовані для кола з центром у точці (1,5; 2,3) і радіусом 3,5. Програма має вивести координати центра кола та його радіус.

[7] Далі побачимо, що в процесі виконання програми змінні можуть додаватися до її пам'яті або, навпаки, вилучатися.

2.7. Поняття вхідного потоку

2.7.1. Поняття потоку

У мові C++ основним поняттям уведення й виведення даних є **потік** – послідовність символів або інших даних. У програмі *потік є представником фізичного файлу* на зовнішньому носії даних (диску, клавіатурі або екрані монітора), а операції обміну даних із файлом зображено як операції добування даних із потоку або дописування їх до нього.

У програмуванні існує поняття **стандартних файлів** уведення й виведення – зазвичай ними є клавіатура та екран. У C++-програмі їм відповідають **стандартний потік уведення** з ім'ям `cin` і **стандартний потік виведення** з ім'ям `cout`. Отже, імена `cin` і `cout`, означені у файлі `iostream`, насправді позначають не клавіатуру та екран, а потоки, що відповідають цим пристроям у програмі. На початку виконання програми обидва потоки `cin` і `cout` порожні.

Із засобів обробки потоків розглянемо лише операції **введення** `>>`, або **добування**, даних зі вхідного потоку, і **виведення** `<<`, або **вставки** (*insertion*), даних у вихідний потік. Розгляд інших засобів виходить за межі цієї книги.

2.7.2. Операція вставки з потоку

Вираз із операцією введення (вставки з потоку) `>>` має вигляд `cin >> ім'я-змінної`. Додавши в кінці виразу символ `;`, маємо інструкцію введення.

Виконуючи операцію введення, комп'ютер забирає зі вхідного потоку послідовність непорожніх символів, за якою створює відповідне значення та присвоює змінній. Усі порожні символи пропускаються. Якщо у вхідному потоці немає непорожніх символів (потік порожній), то комп'ютер очікує на його поповнення. Потік поповнюється, коли людина набирає на клавіатурі деяку послідовність символів (вони з'являються на екрані) і натискає на клавішу `Enter`.

Згідно з типами змінних, в які вводяться значення, операція >> *розбиває вхідний потік на лексеми* (послідовності непорожніх символів) і перетворює їх на значення певних типів (*інтерпретує лексеми*). Отже, операцію >> можна розглядати як *інтерпретацію вхідного потоку символів*.

Значенням виразу введення є той потік, з якого взято символи. До нього знову можна застосувати операцію **>>**, тобто можливі вирази вигляду

`cin >> ім'я-змінної_1 >> ім'я-змінної_2`

та аналогічні їм із більшою кількістю змінних. Наведений вираз еквівалентний виразу

`(cin >> ім'я-змінної_1) >> ім'я-змінної_2`

й задає послідовне введення значень двох змінних.

Приклад. Якщо є змінні **`char c1, c2;`** і під час виконання **`cin >> c1 >> c2;`** натиснути клавіші **`Space`**, **`W`**, **`Space`**, **`Space`**, **`3`**, **`Q`**, **`Enter`**, то змінні отримають значення **`'W'`** та **`'3'`**, а **`Q`** залишиться у вхідному потоці. ◂

За стандартних налаштувань операція >> пропускає всі порожні символи вхідного потоку.

За стандартних налаштувань, утворюючи за вхідною послідовністю числове значення, операція **>>** розглядає вхідну послідовність як *десяткове* зображення числа. Дійсні числа можна задавати в нормалізованій формі, а також без дробової частини. У дробовій частині використовується *крапка, а не кома*. Для цілих чисел дробова частина й порядок не допускаються; задане число повинно належати діапазону можливих значень типу, який має змінна, інакше введення може мати непередбачувані наслідки.

Приклад. Якщо є змінна **`int k;`** і під час виконання **`cin >> k;`** натиснути клавіші **`0`**, **`1`**, **`1`**, **`Enter`**, то значенням **`k`** буде **`11`**. Якщо є змінна **`double x;`** і під час виконання **`cin >> x;`** натиснуто клавіші **`8`**, **`Enter`**, то **`x`** отримує значення **`8.0`**, а якщо **`1`**, **`e`**, **`2`**, **`Enter`**, – значення **`100.0`**. ◂

Під час уведення з клавіатури помилки найчастіше трапляються з числовими змінними. Якщо під час виконання виразу введення сталася помилка, то вхідний потік переходить у **стан помилки**, в якому *виконання подальших операцій уведення неможливе*. За наявності помилки значення змінної, в яку мало відбутися введення, *не змінюється*. Отже, якщо змінну не ініці-

алізовано та значення їй не присвоєно, то її значення залишається "випадковим сміттям"!

Вправи

2.15. Нехай під час виконання програми користувач послідовно натиснув на клавіші `5`, `Enter`, `7`, `Space`, `9`, `Enter`. Що він побачить на екрані?

```
#include <iostream>
using namespace std;
int main() {
   int i, j, k;
   cout<<"Enter 2 integers:"; cin>>i>>j;
   cout<<"Enter an integer:"; cin>>k;
   cout<<"Entered integers: " << i <<
         "," << j << ", " << k <<endl;
   system("pause"); return 0;
}
```

2.16. Чи може змінна `char c1` під час виконання `cin>>c1` отримати значення `' '` (пробіл)?

2.17. Запустіть на виконання програму, наберіть `100e777` і натисніть на `Enter`.

```
#include <iostream>
using namespace std;
int main() {
   int i; double x;
   cout<<"Enter one real:"; cin>>x;
   cout<<"Enter one integer:"; cin>>i;
   cout<<"Entered numbers: "<<x<<", "<<i<<endl;
   system("pause"); return 0;
}
```

Що буде виведено на екран? Чи буде комп'ютер очікувати введення цілого числа? Поясніть отримані результати.

2.18. Що зміниться у виконанні попередньої програми, якщо замість інструкцій `int i; double x;` записати `int i=0; double x=0;`?

Контрольні запитання

2.1. Які обов'язкові частини повинна мати програма мовою С++?

2.2. З яких двох частин складається функція мови С++?

2.3. Які основні групи символів є в алфавіті мови С++?

2.4. Які види лексичних одиниць є в мові C++?
2.5. Що таке ім'я в мові C++?
2.6. Що таке змінна? Що таке змінна в програмуванні?
2.7. Що таке тип даних? Що таке носій і сигнатура типу?
2.8. Що таке константа? Які різновиди констант є в мові C++?
2.9. Що таке вираз та інструкція присвоювання?
2.10. У чому полягає семантика інструкції присвоювання?
2.11. Що таке пам'ять програми та стан пам'яті програми?
2.12. Що є семантикою послідовності інструкцій присвоювання?
2.13. Що таке потік значень?

Задачі

2.1. Напишіть програму, яка виводить повідомлення
`Each algorithm is a list of well-defined instructions for completing a task. Starting from an initial state, the instructions describe a computation that proceeds through a well-defined series of successive states, eventually terminating in a final ending state.`
Зверніть увагу на розташування тексту по рядках. Щоб отримати охайне повідомлення, за необхідності починайте виведення з нового рядка.
2.2. Напишіть програму, яка друкує охайне повідомлення такого змісту: **`Цю програму створив студент … групи … у … році.`** Замість "…" вставте особисті дані (прізвище, ім'я та побаткові, номер групи й рік написання програми).
2.3. Підготуйте реферат на тему: "Способи запису символьних констант і символів у рядках у мові C++".

РОЗДІЛ 3
Прості математичні обчислення

3.1. Арифметичні операції

3.1.1. 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. 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 і значенням 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 C++.

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**. Застосування операцій до різнотипних значень розглянуто нижче.

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** – значення **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.

Інструкція вигляду **cout << вираз;** обчислює та виводить значення виразу.

Вправи

3.1. Що буде виведено на екран за такими інструкціями?

```
cout<<9/5<<' '<<-9/5<<' '<<9/-5<<' '<<-9/-5<<endl;
cout<<9%5<<' '<<-9%5<<' '<<9%-5<<' '<<-9%-5<<endl;
cout<<9./5.<<endl;
```

3.2. Що буде виведено на екран за такими інструкціями?

```
cout<<7/3<<' '<<1/6<<endl;
cout<<7./3.<<' '<<1./6.<<endl;
```

3.3. Припустимо, що значення дійсної змінної **length** відповідає довжині будівлі в міліметрах. Написати вираз, що задає довжину будівлі в метрах.

3.4. Нехай значення цілої змінної **sizeOfFile** задає розмір файлу в байтах. Написати вираз, значенням якого є розмір файлу в Кбайтах.

3.5. Нехай **v** – ім'я цілої змінної з невід'ємним значенням. Написати вираз, який обчислює: а) значення молодшої десяткової цифри числа **v**; б) значення молодшої двійкової цифри числа **v**.

Одномісна операція **sizeof** 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**.

3.1.2. Старшинство операторів і порядок виконання операцій

Мова C++ 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 C++ 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). Докладніше про пріоритети операторів і властивості зв'язування див. у додатку Б.

Приклади

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 $ax^2+bx+c=0$. 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.

Вправи

3.6. Що буде виведено на екран за такими інструкціями?

```
cout<<1+4/2<<' '<<(1+4)/2<<endl;
cout<<2*4%7<<' '<<2*(4%7)<<endl;
cout<<51/6%7<<' '<<51/(6%7)<<endl;
cout<<2*(7%8)<<' '<<12/6%8<<' '<<5-3%2<<endl;
```

3.7. Що буде виведено на екран за такими інструкціями?

```
cout<<4*6/8<<' '<<4/8*6<<endl;
```

3.8. Що буде виведено на екран за такими інструкціями?

```
cout<<(-3+5)*(2%7/3+4*2)<<endl;
```

3.9. Що буде виведено за такими інструкціями?

а)
```
int a=2, b=3;
cout<<"a*b="<<a*b<<";\n***"<<endl;
```

б)
```
int a=3, b; cout<<a*a<<' '<<a+4<<'\n';
```

3.10. Нехай `a`, `b`, `c` – дійсні змінні, що позначають довжини сторін трикутника. Запишіть вираз, що задає периметр, та інструкцію, яка присвоює периметр змінній `p`.

3.1.3. 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

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 $\pi = 3.141593\ldots$ та $e = 2.7182818\ldots$. Натомість у бібліотеці `cmath` означено константи[8] з іменами `M_PI` (число π), `M_PI_2` ($\pi/2$), `M_PI_4` ($\pi/4$), `M_1_PI` ($1/\pi$), `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. Програма

```
#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;
}
```

виводить значення математичних констант π та e. ◂

Бібліотеки систем програмування мовою C++ 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. Докладніше про зазначені й деякі інші функції див. у додатку В.

Функція в мові **C++** – 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. Цей термін буде уточнено в розд. 5.

[8] У деяких версіях мови C++ ці константи замінено відповідними бібліотечними функціями.

Вправи

3.11. Написати вираз мови C++, відповідний до математичного виразу:

а) $\sqrt{a^2+b^2}$; б) $(a+b)^{1/7}$; в) $(a^{12}+b^{12})^{1/3}$; г) $\left|\sqrt{a}-\sqrt{b}\right|$.

3.12. У математиці є тотожність $\log_x y = \ln y/\ln x$. Написати вираз мови C++, відповідний до математичного виразу $\log_x y$.

3.13. Написати програму, що виводить значення $\sqrt{2}$.

3.14. Написати програму, що виводить значення $(2{,}5)^{1{,}6}$.

3.15. Написати послідовні інструкції присвоювання:

а) змінній **d** значення математичного виразу b^2-4ac, а **x1** та **x2** – виразів $(-b-\sqrt{d})/(2a)$ та $(-b+\sqrt{d})/(2a)$;

б) змінній **p** значення математичного виразу (a+b+c)/2, а змінній **s** – виразу $\sqrt{p(p-a)(p-b)(p-c)}$.

3.16. Написати програму виведення на екран чисел π та *e*.

3.1.4. Складені присвоювання

Вирази присвоювання дуже часто мають вигляд, наприклад, **a=a+b, a=a*b** тощо, тобто збільшують значення змінної **a** на величину **b**, множать на **b** тощо. Такі присвоювання можна задавати за допомогою спеціальних операторів у скороченій формі: **a+=b**, **a*=b**, які називають складеними (*compound*) присвоюваннями. На місці **b** можна записати будь-який вираз відповідного типу. Значенням виразу присвоювання є нове значення змінної в його лівій частині. Наприклад, значенням **a+=b** є нове значення змінної **a**.

В описаний спосіб до змінної можна застосувати будь-яку з операцій **+**, **-**, *****, **/**, **%**, а також деякі інші. Наприклад, замість **number=number%10** можна написати **number%=10**, замість **degree=degree+pow(2,n)** – вираз **degree+=pow(2,n)**.

Вправи

3.17. Записати кілька варіантів виразу, що подвоює значення цілої змінної.

3.18. Записати кілька варіантів виразу, що збільшує значення цілої змінної на 2.

3.1.5. Особливості цілих типів

Кожен цілий тип зациклено, тобто наступним після максимального числа в типі є мінімальне (див. с. 13). Нехай у цілому типі є 2^n значень (n – кількість бітів у зображенні даних цього типу) з відповідним максимальним, яке позначимо **MAX**. Якщо сума або різниця двох чисел цілого типу більше **MAX**, то вона зображується в типі числом, яке на 2^n менше справжньої суми чи різниці. Якщо справжня сума або різниця менше мінімального числа цього типу, то результат на 2^n більше її. Аналогічно результатом множення двох чисел деякого типу є деяке число цього самого типу, різниця між яким і справжнім добутком кратна 2^n.

Приклад. У типі **int** значення 12! = 479 001 600 зображується правильно, а 13! = 6 227 020 800 зображується як 1932053504, тобто $13!-2^{32}$. ◂

Вправа 3.19. Що буде виведено на екран за програмою?

```
#include <iostream>
using namespace std;
int main() {
   int iMax=INT_MAX, iMin=INT_MIN;
   cout<<iMax+1<<' '<<iMin-1<<endl;
   system("pause"); return 0;
}
```

3.2. Сумісність і перетворення типів

3.2.1. Сумісність типів за присвоюванням і перетворення типів

Сумісність типів за присвоюванням – це можливість присвоювати змінним одного типу значення іншого.

Усі базові типи мови C++ сумісні за присвоюванням один з одним[9]. У виразі присвоювання змінна отримує значення, перетворене до її типу.

Утворення значення одного типу за значенням іншого називається **перетворенням типу**.

При перетворенні цілого значення до дійсного типу відповідне число зображується в цьому типі. За цим правилом кожне значення типу **`int`** може бути зображене в типі **`double`**. При перетворенні дійсного значення до типу цілих у ньому відкидається дробова частина. Якщо ціла частина, що залишилася, не належить діапазону чисел типу **`int`**, то результат перетворення буде помилковим.

Приклади

1. Якщо змінній **`double x`** присвоюється **`1`**, то її значенням стає дійсне **`1.0`**.

2. Якщо змінній **`int a`** присвоюється **`1.9`**, то її значенням стає **`1`**, а якщо **`-1.9`**, то **`-1`**.

3. Нехай є такі змінні та присвоювання:

```
double dd=3.5e38; int id=dd;
```

Змінні отримають значення: **`dd=3.5e38, id=-2147483648.`** ◀

Присвоюючи вираз одного типу змінній іншого типу, контролюйте, щоб значення виразу можна було зобразити в типі змінної без втрат. Уникайте присвоювань дійсних виразів змінним цілих типів, оскільки це може мати *непередбачувані наслідки*.

[9] У мовах програмування не всі типи сумісні за присвоюванням. Наприклад, є мови, що не дозволяють дійсне чи символьне значення присвоїти цілій змінній, а ціле значення – символьній. Проте більшість мов дозволяють присвоювати цілі значення дійсним змінним.

За інструкцією `int i=UINT_MAX;` змінна `i` отримає значення `-1`. Не наводячи повного переліку правил перетворень типів, зазначимо лише, що константа `UINT_MAX` задає найбільше беззнакове ціле число 4294967295, а воно не може бути зображено в типі `int`.

Арифметичний тип вважається **більшим** за інший арифметичний тип, якщо для зображення його значень потрібно більше байтів або він має більше максимальне значення.

Уникайте присвоювань значень більшого цілого типу змінним меншого цілого типу, оскільки це може мати *непередбачувані наслідки*. Зокрема, уникайте перетворення значень беззнакових цілих типів на значення знакових типів.

При перетворенні значення типу `char` до типу `int` однобайтове зображення символу розглядається як зображення[10] числа. При зворотному перетворенні числа від 0 до 127 (а за стандартних налаштувань компілятора й чисел від -128 до -1) до типу `char` результатом є символ, зображений в одному байті так само, як і це число. Наприклад, числу 32 відповідає `' '` (пробіл), числу 48 – символ `'0'`, `49 - '1'` тощо, числу 65 – символ `'A'`, 66 – `'B'` тощо, числу 97 – `'a'`, 98 – `'b'` тощо.

Логічне значення `false` перетворюється на ціле число `0`, а `true` – на `1`. Нульове значення будь-якого числового типу перетворюється на `false`, а **всі** ненульові значення – на `true`.

Вправа 3.20. Яке значення отримає змінна `c` за інструкцією `char c=0;` ?

3.2.2. Сумісність і правила перетворення типів у виразах

Сумісністю відмінних між собою типів у виразах називається можливість сумісного використання операндів цих типів.

Типи дійсних і цілих чисел є сумісними, тобто у виразах можна записувати операнди цих різних типів, наприклад, `1+2.0`.

[10] Залежно від налаштувань компілятора воно може бути знаковим (за стандартних налаштувань) або беззнаковим.

Якщо один з операндів цілий, а другий дійсний, то за цілим значенням утворюється відповідне дійсне, і операція застосовується до дійсних значень.

Отже, за виконання `1+2.0` спочатку ціле `1` перетворюється на дійсне `1.0`, а потім додаються дійсні числа.

Усі цілі (*integral*) типи також є сумісними у виразах. Зокрема, при обчисленні виразів значення типів `char` та `bool` спочатку завжди перетворюються на значення типу `int`. Тому вирази `'5'-'1'` та `'F'-'A'` є синтаксично правильними й мають значення 4 та 5, відповідно.

Вправи

3.21. Що буде виведено на екран за програмою? Зверніть увагу на відмінність ділення цілих і дійсних чисел.

```
#include <iostream>
using namespace std;
int main() {
   cout<<(3/5)*20+32.<<' '<<(3/5.)*20+32<<endl;
   system("pause"); return 0;
}
```

3.22. Яке значення отримає змінна `c` за інструкцією `char c='f'-'b';` ?

3.2.3. Явне перетворення типів

Перетворення типів значень, описані вище, відбуваються без явних на це вказівок, тобто неявно. Перетворення можна задати *явно* у вигляді ***тип*`(`*вираз*`)`** або **`(`*тип*`)`*вираз***. Пріоритет перетворення такий самий як і в інших префіксних одномісних операторів.

Приклади

1. Вираз `int(2.8)` має значення 2, `double(-2)/3` – значення `-0.666`…, а `double(-2/3)` – значення `0.0`.

2. Значенням виразу `char(48)` є символ `'0'`, а виразу `char('0'+7)` – символ `'7'`.

3. У виразах часто зустрічаються дійсні значення дробів із цілими чисельником і знаменником. Якщо не перетворити один із цих операндів до дійсного типу, то значення виразу відрізнятиметься від потрібного. Наприклад, за будь-якого додатного

значення цілої змінної **n** дріб **n/(n+1)** має значення **0**; щоб отримати потрібне дійсне значення, необхідно явно перетворити тип: **(double)n/(n+1)**, **double(n)/(n+1)** або **n/double(n+1)**. ◂

3.3. Виведення значень виразів

3.3.1. Операція вставки у вихідний потік

Вираз із операцією виведення (вставки) у вихідний потік **<<** має вигляд **cout << *вираз***. Додавши в кінці виразу символ **;**, отримаємо інструкцію виведення.

Операція **<<** виконується так. Обчислюється значення виразу праворуч, за ним утворюється послідовність символів, яка зображує це значення, і виводиться в потік (за **cout** – на екран). Результатом операції є потік, і до нього можна знову застосувати операцію **<<**. Це уможливлює вирази вигляду

cout << *вираз_1* << *вираз_2*

та аналогічні з більшою кількістю виразів. Коли послідовно виводяться два вирази, пробіли між ними не додаються. За стандартних налаштувань числа виводяться в десятковому записі.

Приклад. Після виконання інструкцій

```
int a=44; double b=1.2e-2;
cout << "a=" << a << ";\n[" << "b=" << b << ']';
```

на екран виводяться такі рядки:

```
a=44;
[b=0.012]
```

З константи **";\n["** виводиться символ **;**, керувальний символ **\n** задає перехід курсора на новий рядок, в якому першим є символ **[**. ◂

Пріоритет оператора **<<** нижчий за пріоритет **+** та **–** і вищий за пріоритет порівнянь (див. додаток Б).

Порядок обчислення виразів-операндів **<<** залежить від компілятора. Більшість компіляторів будують машинний код, в якому вирази обчислюються у зворотному порядку, тобто *від останнього до першого*. Це важливо, коли вирази змінюють значення змінних.

Приклад. Після виконання інструкцій

```
int k=3; cout << k*11 << ' ' << (k=7) << '\n';
```

на екран виводяться символи `77 7`, оскільки спочатку значенням `k` стає `7`, а потім, уже за нового значення `k`, обчислюється `k*11`. ◂

Виразів виведення, що змінюють значення змінних, краще *уникати*. Наприклад, інструкцію виведення з наведеного прикладу краще замінити такими:

```
k=7; cout << k*11 << ' ' << k << '\n';
```

Якщо ж спочатку треба надрукувати старе значення змінної `k`, помножене на `11`, а потім її нове значення, то слід писати так.

```
cout << k*11 << ' '; k=7; cout << k << '\n';
```

3.3.2. Елементи форматування вихідного потоку

Бібліотека `iostream` містить засоби, що дозволяють керувати процесами введення й виведення. Розглянемо деякі з них, що дозволяють керувати формою зображення дійсних значень, шириною поля та способом притискання до його країв.

Формати дійсних значень. Інструкція

```
cout << 1.0 << ' ' << 0.1 << ' ' << 0.00000001;
```

виведе на екран `1 0.1 1e-008`. Перше число виглядає як ціле, друге має дробову частину, третє – від'ємний порядок. Чому формати (або форми) виведення чисел такі різні та як ними керувати?

Дійсні значення виводяться або з **фіксованою** (*fixed*) **крапкою**, або в **нормалізованій** (**науковій**) **формі**. Після інструкції `cout<<scientific;` дійсні числа виводяться в нормалізованій формі, а після `cout<<fixed;` – з фіксованою крапкою. Для обох форматів інструкція вигляду `cout.precision(2);` задає кількість знаків дробової частини (тут їх 2). Якщо спосіб виведення явно не встановлено, то засоби виведення обирають його самі.

Приклад. Після виконання програми

```
#include <iostream>
using namespace std;
int main() {
   cout<<scientific; cout.precision(1);
   cout << 1.14 << "  " << 0.16 << '\n';
   cout<<fixed; cout.precision(1);
   cout << 1.14 << "  " << 0.16 << '\n';
   system("pause"); return 0;
}
```

prog005.cpp

отримаємо результат

```
1.1e+000  1.6e-001
1.1  0.2
```

Зверніть увагу: відбулося *округлення* дійсних чисел до вказаної кількості дробових знаків. ◂

Вправи

3.23. Що буде виведено за інструкціями?

```
cout << fixed; cout.precision(2);
cout << -123.321 << ' ' <<345.2 << ' ' <<
        11.777 << ' ' << 13. << ' ' << 7;
```

3.24. Що буде надруковано за інструкціями?

```
cout<<scientific; cout.precision(2);
cout<<12.151; cout<<" 12.151"; cout<<'\n';
cout<<fixed; cout.precision(2);
cout<<12.151; cout<<" 12.151"; cout<<'\n';
```

3.25. Написати програму, що виводить значення $\sqrt{2}$ з вісьмома знаками після десяткової крапки.

3.26. Написати програму, що виводить значення $(2{,}5)^{1{,}6}$ із чотирма знаками після десяткової крапки.

Інтерпретація символів. Операція `<<` перетворює значення певного типу на послідовність символів, яка зображує це значення, і виводить її у вихідний потік. При цьому *вивідні символи інтерпретуються*, тобто виконуються особливі дії, задані деякими символами, наприклад, за появи керувального символу `'\n'` виведення продовжується в новому рядку. Саме інтерпретація символів дозволяє виводити одні й ті самі значення в різних форматах, як, наприклад, дійсні числа з різною кількістю десяткових знаків.

Кінець рядка можна позначити іменем `endl` (скорочення від *end of line* – кінець рядка): вираз `cout<<endl`, як і `cout<<'\n'`, задає виведення символу `'\n'` у вихідний потік. Насправді він означає дещо більше, але розгляд цього виходить за межі посібника.

Ширина поля та притискання до його краю. Виклик функції `width` дозволяє задати певну *ширину поля*, яке займуть символи значення, що буде виводитися наступним. Наприклад, інструкція `cout.width(10);` установлює ширину поля 10. Якщо для наступного значення потрібно більше символів, ніж визначає задана

ширина, то друкуються всі символи, а якщо менше, то спочатку виводяться пробіли (доповнюючи кількість символів значення до заданої ширини), а потім – символи значення. Установлена ширина стосується лише одного елемента виведення.

Приклад. За виконання інструкцій

```
cout<<'#'; cout.width(4); cout<<12<<'#';
cout<<'#'; cout.width(4); cout<<123456<<'#';
```

на екрані з'явиться текст `#  12##123456#`. Тут виведено два пробіли перед `12` і всі цифри `123456`. На символи `'#'`, тобто елементи, що йдуть за числовими константами, установлена ширина поля не впливає. ◂

За встановленої ширини поля вивідні символи притискаються до правого краю поля. Притискання до лівого краю можна задати інструкцією `cout<<left;` . Її дія поширюється на всі подальші виведення, в яких установлено ширину поля, поки не буде задано притискання до правого краю, тобто `cout<<right;` .

Вправи

3.27. Написати програму, що запитує в користувача координати точки дійсної площини, а потім виводить отриману точку. Зверніть увагу на зовнішній вигляд друкованого результату.

3.28. Написати програму, що запитує в користувача коефіцієнти рівняння $ax^2+bx+c=0$, а потім виводить отримане рівняння (x^2 можна вивести у вигляді `x^2`). Зверніть увагу на зовнішній вигляд друкованого рівняння.

3.4. Приклад програми й поняття якості коду

3.4.1. Коментарі та якість коду

Напишемо програму, що вводить температуру за Цельсієм (ціле число) і виводить температуру за Кельвіном, більшу на

273 (273 – константа Кельвіна). Розглянемо початковий (не найкращий!) варіант.

```
/* Програма вводить температуру за Цельсієм і
   виводить температуру за Кельвіном */
#include <iostream>
using namespace std;
int main()
{
  int tC,          // температура за Цельсієм
      tK;          // температура за Кельвіном
  cout << "Celsium to Kelvin.\n";
  cout << "Enter temperature by Celsium " <<
          "(int>=-273):";
  cin >> tC;
  tK = tC+273;
  cout << "temperature by Kelvin: " << tK << endl;
  system("pause"); return 0;
}
```

Порівняно з попередніми прикладами ця програма має нові елементи. У першому й другому рядках записано **коментар** – послідовність символів між парами символів `/*` та `*/`, яка не містить `*/`. Такий коментар може займати кілька рядків і бути записаним між будь-якими двома лексемами. Інші коментарі є послідовностями символів, що починаються парою символів `//` і закінчуються в кінці рядка, який їх містить. Під час компіляції коментарі пропускаються.

Програми можна оцінювати за якістю, причому з різних позицій. На прикладі обчислення температури за Кельвіном розглянемо якість програми з погляду легкості її розуміння, використання й модифікації. Розглянемо ще один варіант програми.

```
#include <iostream>
using namespace std;
int main()
{  int tC, tK;
   cin >> tC;
   tK = tC+273;
   cout << "temperature by Kelvin: " << tK << endl;
   system("pause"); return 0;
}
```

Обидві програми розв'язують одну й ту саму задачу, але перша краща тим, що зрозуміти її набагато легше. Отже, *код пови-*

нен читатися та сприйматися як розмовна мова. Мартін Фаулер сформулював це так:

"Кожен дурень може написати код, зрозумілий комп'ютеру. Добрі програмісти пишуть код, зрозумілий людям."

Чому код *повинен* бути зрозумілим людині? Справа в тому, що код, який у сучасних програмах вимірюється мільйонами (!) рядків, необхідно **обслуговувати** – налагоджувати, виправляти помилки й модифікувати в разі змін вимог до нього. І робити все це має не комп'ютер, а людина.

Щоб полегшити людині роботу з кодом, можна використати коментарі. Однак ними не слід зловживати. Коментар біля кожної інструкції програми не зробить код зрозумілішим. Високоякісний код має містити коментарі тільки там, де вони дійсно *необхідні*.

Зміст коментарів може бути довільним, але зазвичай у коментарях зазначають:

– що повинна виконувати програма, функція чи фрагмент коду;

– яку сутність моделює або для чого використовується змінна;

– умови, що мають справджуватися *перед* виконанням певних інструкцій, щоб забезпечити їх коректність (так звані **pre-умови**, або **передумови**);

– умови, що справджуються *після* виконання певних інструкцій (**post-умови**, або **післяумови**);

– поведінку програми за некоректних даних.

3.4.2. Змістовні імена

Отже, одним із засобів прояснити призначення коду й полегшити його сприйняття є коментарі. Проте для ефективного обслуговування коду їх замало. Якщо код необхідно виправити або модифікувати, то все рівно доведеться розібратися в ньому до дрібниць.

Код має бути зрозумілим *навіть без додаткових коментарів*. (Звісно, про складні математичні алгоритми тут не йдеться.)

На практиці, якщо код стає дуже заплутаним і незрозумілим, то для виправлення навіть однієї помилки в його логіці буває простіше переписати все з чистого аркуша, ніж внести зміни в існуючий код. І коментарі тут не врятують!

Щоб уникнути зайвих коментарів, варто обирати імена, що відображають призначення відповідних змінних і функцій. У прикладі є коментарі щодо призначення змінних `tC` та `tK`. Програма коротенька, і побачити коментарі легко. Проте за більшої програми довелося б шукати відповідні пояснення серед тисяч рядків коду. Отже, краще не коментувати зміст змінних, а дати їм *змістовні імена*, що відображають їх призначення. Замінимо імена `tC` та `tK` іменами `tCelsium` та `tKelvin`, відповідно, де літера `t` є скороченням від `temperature` (температура). Зауважимо: у бібліотеках сучасних систем програмування використовуються саме змістовні імена (`cin`, `cout`, `pow`, `sqrt` тощо), і це істотно полегшує сприйняття коду.

3.4.3. Іменування констант

Ще один недолік нашої програми – використання константи `273`. Такі "магічні" константи можуть з'являтися в різних місцях програми, але людина може помилитися, і десь набрати не `273`, а, скажімо, `237`. Синтаксично програма залишиться правильною, а семантично – ні. Ще одна проблема з "магічними" константами виникає, коли в процесі створення програми таку константу потрібно змінити. Адже в усій програмі вона має змінитися *однаково*!

Одним із засобів уникнути "магічних" констант є директива `#define` (означити). У найпростішому випадку її записують так:

```
#define ім'я значення
```

Після такої директиви всі лексеми вхідної програми, що збігаються з указаним іменем, препроцесор замінює на вказане значення. Значенням може бути довільна послідовність символів. Наприклад, за наявності директиви

```
#define ZERO 0
```

інструкція `int i=ZERO;` перетвориться препроцесором на `int i=0;`, а інструкції `int ZERO1;` та `cout<<"ZERO";` залишаться без змін, оскільки лексемами в них є `ZERO1` та `"ZERO"`, а не `ZERO`.

Якщо в директиві `#define` вказано тільки ім'я, а значення відсутнє, то препроцесор вважає, що імені відповідає порожня послідовність символів.

Можна вважати, що директива **#define** дає константі ім'я, тобто *іменує константу*.

За наявності директиви **#define KELVIN 273** замість "магічної" константи 273 можна використовувати іменовану константу з ім'ям **KELVIN** і значенням **273**. Отже, програма стане такою:

```
/* Програма вводить температуру за Цельсієм і */
/*   виводить температуру за Кельвіном        */
#include <iostream>
using namespace std;
#define KELVIN 273
int main()
{ int tCelsium,tKelvin;
  cout << "Celsium to Kelvin.\n";
  cout << "Enter temperature by Celsium (int>=-"
       <<  KELVIN << "):";
  cin >> tCelsium;
  tKelvin = tCelsium+KELVIN;
  cout << "temperature by Kelvin: " << tKelvin << endl;
  system("pause"); return 0;
}
```

prog006.cpp

◀

3.4.4. Константи як змінні з незмінним значенням

Директива **#define** має кілька недоліків, з яких розглянемо два. По-перше, у директиві як значення можна записати будь-яку послідовність символів, тому через необережний запис директиви препроцесор може змінити лексичну структуру програми. По-друге, тип значення не задано явно, тому його визначає компілятор, а це не завжди бажано. Уникнути цих недоліків можна, використовуючи **змінні з незмінюваним значенням**.

Зарезервоване слово **const** (його називають **специфікатором типу**) перед іменем типу в інструкції оголошення імені змінної означає, що значення цієї змінної не можна модифікувати. Змінна ініціалізується в оголошенні значенням будь-якого виразу відповідного типу (якщо вираз містить ім'я іншої змінної, то їй раніше має бути присвоєно значення). Запис її імені ліворуч в інструкції присвоювання є помилкою.

Приклади

1. У програмі про температуру замість директиви `#define KELVIN 273` можна записати зовнішнє, тобто розташоване *зовні головної функції*, оголошення імені:

```
const int KELVIN=273;
```

Оголошення задає ім'я цілої змінної зі значенням `273`. Її можна використовувати у функціях, записаних нижче в програмі. Решта програми залишається без змін.

2. Розглянемо інструкції оголошення

```
int good=8, bad;
const int good4=4*good, bad4=4*bad;
```

Змінна `good4` отримає значення `32`, яке в подальшому не можна змінити, а значення `bad4` непередбачуване, оскільки перед її ініціалізацією змінна `bad` значення не отримала. ◂

Змінні, оголошені зі словом `const`, часто використовують для іменування констант. В іменах констант прийнято (хоча й не обов'язково) використовувати тільки великі літери.

Корисно іменувати вирази з константним значенням, особливо якщо вони потрібні в багатьох місцях програми. У цих місцях замість виразу достатньо записати лише його ім'я. Окрім того, за необхідності змінити вираз достатньо зробити це лише в іменуванні, а якщо вираз не іменовано, то доведеться змінювати його всюди, де він зустрічається.

Контрольні запитання

3.1. Що є семантикою арифметичного виразу?
3.2. Яка математична операція, оператор якої відсутній у мові C++, має властивість правобічного зв'язування?
3.3. Який вигляд має виклик функції?
3.4. Що таке сумісність типів?
3.5. Що таке перетворення типу?
3.6. Що таке сумісність типів за присвоюванням?
3.7. Як виглядає явне перетворення типу?
3.8. Чи кожне дійсне число можна точно зобразити в типі `double`?