### 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 С++.

Операції **додавання**, **віднімання**, **множення** й **ділення** мають знаки відповідно **+, -, \*, /**. Результатом операції з *цілими* числами є *ціле* число, з *дійсними* – *дійсне*. Наприклад, значенням виразу **4/2** є ціле число **2**, а виразу **4.0/2.0** – дійсне **2.0**. 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 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). 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**(-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  3.141593… та *e* 2.7182818…. Натомість у бібліотеці **cmath** означено константи[[1]](#footnote-1) з іменами

**M\_PI** (число ), **M\_PI\_2** (/2), **M\_PI\_4** (/4), **M\_1\_PI** (1/), **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  та *e*. 

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. Докладніше про зазначені й деякі інші функції див. у додатку В.

**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. Цей термін буде уточнено в розд. 5.

#### Вправи

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

а) *a*2 *b*2 ; б) (*a* *b*)1/7 ; в) (*a*12  *b*12 )1/3; г) *a* *b* .

3.12. У математиці є тотожність log*x y*  ln *y*/ln *x*. Написати вираз мови С++, відповідний до математичного виразу log*x y* .

3.13. Написати програму, що виводить значення 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*2-4*ac*, а **x1** та **x2** – виразів (*b*  *d* )/(2*a*) та (*b*  *d* )/(2*a*);

б) змінній **p** значення математичного виразу (*a*+*b*+*c*)/2, а змінній **s** – виразу *p*(*p*  *a*)(*p* *b*)(*p*  *c*) .

3.16. Написати програму виведення на екран чисел  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! 479001600 зображується правильно, а 13! 6227020800 зображується як 1932053504, тобто 13!232. 

**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 С++ сумісні за присвоюванням один з одним[[2]](#footnote-2). 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.

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[[3]](#footnote-3) 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**.

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**.

1. Значенням виразу **char(48)** є символ **'0'**, а виразу **char('0'+7)** – символ **'7'**.
2. 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 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 повинен 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 **preумови**, 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 будьяку 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.

1. 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.

1. У деяких версіях мови C++ ці константи замінено відповідними бібліотечними функціями. [↑](#footnote-ref-1)
2. 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. [↑](#footnote-ref-2)
3. Залежно від налаштувань компілятора воно може бути знаковим (за стандартних налаштувань) або беззнаковим. [↑](#footnote-ref-3)