Лекція 2. Оператори мови С, обчислення виразів

1. Операції над даними цілих типів.

2. Операції над числовими даними дійсних типів.

3. Логічні операції.

4. Спеціальні форми оператора присвоєння.

1. Операції над даними цілих типів

З цілочисельними даними (константами и змінними) в мовах С, С++ можна виконувати звичайні арифметичні операції: додавання (x+y), віднімання (z-5), множення (x1*x3) и ділення (y/w). На відміну від мови Pascal тут ділення цілочисельних операндів дає цілочисельний результат. Наприклад, 5/2 = 2. Для обчислення остачі від ділення цілих чисел в С/С++ використовується операція %, наприклад, 5%2=1.

Над цілочисельними даними можна також виконувати операції порозрядного зсуву (знаки «<<» та «>>»):

```
y=x<<3; // зсув вліво на три двійкові розряди
z=y>>5; // зсув вправо на п'ять двійкових розрядів
```

Операція зсуву вправо по-різному працює для цілих чисел зі знаком і цілих чисел без знака. Наприклад:

```
#include <stdio.h>


int main()
{
  int x=5, y=-5;
  unsigned z=0xFFFFFFFB;
  printf("x=%x y=%x z=%x",x,y,z);
  printf("\nx<<2=%x x>>2=%x",x<<2,x>>2);
  printf("\ny<<2=%x y>>2=%x",y<<2,y>>2);
  printf("\nz<<2=%x z>>2=%x",z<<2,z>>2);
  return 0;
}
```

```
//=== Результат роботи ===
x=5 y=fffffffb z=fffffffb
x<<2=20 x>>2=1
y<<2=ffffffec y>>2=fffffffe
z<<2=ffffffec z>>2=3ffffffe
```

Для виводу результатів роботи програми в прикладі вище викликається бібліотечна функція ***printf()***. Першим параметром їй передається так звана "форматна стрічка", – вона складається з тексту, що виводиться на екран з позначеними "місцями" для вставки значень наступних параметрів функції. Такі позначки називають форматними вказівниками, вони починаються знаком відсотка і завершуються літерою, що вказує формат виводу даних. В нашому прикладі місця, куди виводяться значення змінних та результатів операцій позначені форматним вказівником "**%x**", який вказує на вивід цілих чисел у шістнадцятковому представленні (системі числення з основою 16).

Над однойменними двійковими розрядами цілочисельних операндів можна також виконувати логічні операції - логічне додавання «|», логічне множення «&», виключне або «^» та інвертування «~». В літературі за цими операції закріпилася назва «побітові». Розглянемо їх дію на прикладі програми (для того, щоб правильно інтерпретувати результат її роботи слід врахувати, що число 5 має двійковий код 00...0101, а 7 - двійковий код 00...0111):

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int x=5, y=7;
  printf("x=%x y=%x",x,y);
  printf("\nx|y=%x x&y=%x",x|y,x&y);
  printf("\nx^y=%x ~x=%x",x^y,~x);
  getch();
  return 0;
}
//=== Результат роботи ===
x=5 y=7
x|y=7 x&y=5
x^y=2 ~x=fffffffa
```

При обробці цілочисельних даних можна використовувати системні функції математичної бібліотеки. Прототипи цих функцій описані в заголовних файлах ***math.h*** і ***stdlib.h***. Згадаємо деякі з них.

Функція ***abs(x)*** повертає модуль свого аргументу. Аргументом функцій ***atoi(s)*** і ***atol(s)*** є рядок, що представляє запис цілого числа. Кожна з цих функцій перетворює символьний запис числа у відповідний машинний формат (результат atoi має тип int, результат atol – тип long) і повертає отриманий результат. Функції ***itoa*** і ***ltoa*** дозволяють отримати текстовий запис цілого числа у вказаній системі числення. Перший їхній аргумент – числове значення типу int або long. Другим аргументом є символьний масив (або вказівник на рядок символів), куди записується результат перетворення. А третій аргумент, значення якого знаходиться в діапазоні від 2 до 36, визначає підставу системи числення, в яку перетвориться значення першого аргументу.

У ряді математичних алгоритмів, що використовують ймовірнісні методи, а частіше – в ігрових програмах активно використовуються різні генератори випадкових чисел. Функція ***rand()*** при кожному повторному зверненні до неї видає чергове випадкове число з діапазону від 0 до ***RAND_MAX*** так, щоб ці числа мали рівномірний розподіл за ймовірністю. Значення ***RAND_MAX*** може залежати від системи та реалізації бібліотек, в конфігурації системи, в якій виконувався код прикладів посібника RAND_MAX = 32767. Якщо ж потрібно випадковим чином отримати ціле число з вказаного діапазону від n1 до n2, то можна задіяти конструкцію на кшталт:

```
int random_number = n1 + rand() % (n2 - n1 + 1);
```

Щоправда, функція rand() може генерувати одне і те ж число, оскільки її "випадковість" прив'язана до часу виконання програми, який за тих самих умов може виявитися практично сталим. Для усунення цієї проблеми можна використати функцію "збудження" генератора випадкових чисел ***srand()***, передавши їй значення часу за системним годинником ПК:

```
srand(time(0));

int random_number = -10 + rand() % (21);
```

Цей код буде при кожному запуску програми визначати довільне ціле число відрізка [-10; 10].

2. Операції над числовими даними дійсних типів

Для обчислень з дійсними даними дозволяється використовувати лише чотири арифметичні операції - додавання (+), віднімання (-), множення (*) і ділення (/). В арифметичному виразі можуть зустрічатися операнди різних типів, тому важливо знати, як компілятор визначає тип результату. Арифметичні дії виконуються за такими правилами:

- спочатку виконуються дії у внутрішніх дужках;
- всередині дужок порядок дій визначається пріоритетом операцій - спочатку одномісні операції типу зміни знака, інвертування і

обчислення функцій, потім множення і ділення, в останню чергу, додавання і віднімання;

- тип результату функції визначений в її описі;
- тип результату добутку визначається типом «найширшого» множника;
- тип результату суми, визначається типом «найширшого» доданка.

Попри простоту цих правил, результати обчислення деяких виразів можуть завдати клопотів не дуже уважним програмістам. Наприклад, тип виразу 5/2 +1.0 повинен бути дійсним, бо компілятор орієнтується на тип числової константи 1.0, яка за правилами системи має тип double. Але результат обчислення даного виразу дорівнює 3.0, бо тип кожного доданка формули визначається незалежно від типів інших доданків. Перший доданок є часткою двох цілих констант, тому його тип теж цілий, тобто результат ділення дорівнює 2, а не 2,5. Далі значення обидвох доданків приводяться до типу double, і підсумковий результат 2.0 + 1.0 = 3.0.

Для обчислення математичних виразів бібліотеки C++ надають різноманітні математичні функції. Список деяких з них:

Прототип функції	Результат
`acos( x )`	`arccos x`
`asin( x )`	`arcsin x`
`atan( x )`	`arctg x`
`ceil( x )`	`заокруглення "зверху"`
`cos( x )`	`cos x`
`cosh( x )`	`ch x`
`exp( x )`	e^x
`fabs( x )`	`|x|`
`floor( x )`	`заокруглення "знизу"`
`log( x )`	`ln x`
`log10( x )`	`lg x`
`pow( x, y )`	X^y
`sin( x)`	`sin x`
`sinh( x)`	`sh x`
`sqrt( x)`	`квадратний корінь з x`
`tan( x)`	`tg x`
`tanh( x)`	`th x`

3. Логічні операції

Програмування не зводиться до виконання виключно математичних обчислень з числовими даними, – часто програми виконують різні дії на основі прийнятих внаслідок порівняння даних рішень. В такому випадку говорять про логічні дані, логічні значення та логічні операції. Такі дані можуть приймати лише два значення: або значення true (істина), або значення false (хибність). Такий тип даних ще називають булевим, від прізвища англійського математика Джорджа Буля, з іменем якого тісно пов'язана розробка основ математичної логіки. Окремий тип bool для зберігання логічних даних з'явився в пізніших стандартах С++, а у мові С, та початкових версіях С++ для цього використовувалися числові дані, що інтерпретувалися за правилом: ***0=false***, ненульове значення – ***true***. При цьому ***true*** в числовому вигляді передавалося як 1. В пізніших версіях С++ таке ж правило використовується для приведення даних з логічного типу в число і навпаки. Наприклад, наступний фрагмент коду:

```
int x = 10;

int n = x < 80;

bool b = x;

cout << "Логічне в ціле 10 < 80 = "<< n <<"\n";

cout << "Ціле в логічне (bool)10 = "<< b <<"\n";
```

виведе на консоль:

```
Логічне в ціле 10 < 80 = 1;

Ціле в логічне (bool)10 = 1;
```

Для отримання і опрацювання логічних даних в С/С++ використовуються дві групи операторів: оператори відношення та логічні оператори.

Оператори відношення

СИМВОЛ	ТВЕРДЖЕННЯ
==	Лівий операнд дорівнює правому
!=	Лівий операнд не дорівнює правому
<	Лівий операнд менший за правий
>	Лівий операнд більший за правий
<=	Лівий операнд менший або дорівнює правому
>=	Лівий операнд більший або дорівнює правому

Наприклад:

```
cout << (5> 3);//Виведе 1, бо твердження (5>3) істинне.

cout << (3 <2); //Виведе 0, бо (3<2) хибне.

cout<<(5!=3); // 1, бо «5 не дорівнює 3» - істина.

cout<<(3==2); // 0, бо «3 дорівнює 2» хибність.
```

Часто умови в прикладних задачах задаються не кількома операціями порівняння. Наприклад, щоб перевірити чи знаходиться число в діапазоні від 1 до 10, необхідно перевірити два твердження: число повинне бути одночасно більшим або рівним (>=)1 та меншим або дорівнювати (<=) 10. Для того щоб реалізувати таку комбінацію використовують логічні операції.

Логічне І (знак &&) об'єднує два твердження і повертає істину тільки в тому випадку, якщо і ліве і праве твердження істинні. Розглянемо її дію на прикладі, в якому програма перевіряє, чи введене ціле число належить відрізку [1; 10].

```
#include <iostream>

using namespace std;

void main()

{

     int N;

     cout<<"Enter digit:\n";

     cin>>N;

     cout<<((N>=1)&&(N<=10));

     cout<<"\nIf you see 1, "

          <<"your digit is in [1; 10]\n\n";

     cout<<"\nIf you see 0, "

          <<"your digit is not in [1; 10]\n\n";

}
```

Логічне АБО (знак ||) об'єднує два твердження і повертає істину у випадку, якщо хоча б одна з частин істинна. Логічне НЕ (знак !) є унарною операцією і використовується тоді, коли потрібно змінити результат перевірки на протилежний.

Результат дії логічних операцій подано в таблиці

A	B	!A	A && B	A || B
true	True	false	True	true
true	False	false	False	true
false	True	true	False	true
false	False	true	False	false

Несподіване перетворення числових даних в логічні деколи може призводити до непередбачуваних помилок в роботі програми, які важко відслідкувати. Найчастіше вони виникають через неправильне порівняння двох значень на рівність. На місці оператора відношення "дорівнює" який записується за допомогою двох послідовних знаків "= =" через неуважність автора коду може

опинитися оператор присвоєння (один знак "="). Оскільки оператор присвоєння, окрім запису даних в змінну, ще й повертає присвоєне значення в місце свого виклику, то в місці перевірки умови замість логічного отримується деяке числове значення. Далі це числове значення конвертується в логічне за наведеним вище правилом. Так, наприклад, наступний код

```
cout<<"Enter digit:\n";

cin>>N;

if(N = 100) cout<<"N = 100\n";

else cout << "N > 100 or N < 100\n";
```

завжди друкуватиме

```
N = 100
```

незалежно від того, яке саме число в N введе користувач.

4. Спеціальні форми оператора присвоєння

Часто в програмах виникає потреба виконати якусь дію для послідовних даних (перевірити послідовність цілих чисел наявність певної властивості, переглянути послідовно усі символи деякого тексту чи елементи з деякого набору). У таких випадках важко обійтися без операцій ***інкременту*** (присвоєння змінній значення на 1 більшого за її попереднє значення) та ***декременту*** (присвоєння значення на 1 меншого за попереднє значення). Такі операції полягають в отриманні значення змінної, обчисленні та присвоєнні їй її нового значення. Їх можна задати за допомогою операторів присвоєння вигляду ***x=x+1*** чи ***x=x-1***, проте в C/C++ (та їх численних нащадках) існують більш лаконічні форми:

```
x++;// замість x=x+1;

x--;// замість x=x-1;

++x;// замість x=x+1;

--x;// замість x=x-1;
```

У наведених прикладах розміщення подвійного знака плюс чи мінус ролі не відіграє, бо стосується лише зміни самої змінної. Проте така конструкція може бути частиною арифметичного виразу, чи, наприклад виконувати роль індексу елемента масиву:

```
y=a[i++];

z=b[++j];
```

У першому випадку змінній ***y*** буде присвоєно значення a[i] зі старим індексом, після чого індекс i буде збільшений на 1 (в цьому випадку говорять про ***постфіксну форму інкремента***). У другому випадку спочатку індекс j буде збільшений на 1, а потім значення b[j] з новим індексом братиме участь

в операції присвоєння, цей випадок називають ***префіксною формою інкремента***. Аналогічною є дія ***префіксної*** та ***постфіксної*** форм ***декремента***.

Ще одна форма оператора присвоювання походить від умовного арифметичного виразу, що вперше з'явився в мові АЛГОЛ-60:

```
v = a > b? e1 : e2;
```

Насправді в записі вище фігурують два оператори: так званий ***тернарний оператор*** справа від знака "=", та власне сам оператор присвоєння. Виконується такий код наступним чином: перевіряється умова, записана в круглих дужках перед знаком питання, якщо вона задовольняється, то змінній v присвоюється значення виразу e1, в протилежному випадку в змінну v записується значення виразу e2. Дія тернарного оператора подібна до дії функції розгалуження в електронних таблицях. Його можна використовувати не тільки в якості правої частини оператора присвоєння, але і як елемент виводу, чи частину більшого арифметичного виразу.

Наступна група операторів присвоювання зобов'язана своїм походженням формату машинних команд в двоадресних машинах:

```
COD A1,A2
```

тут ***COD*** – код машинної операції; ***A1, A2*** – адреси комірок пам'яті.

Результат виконання такої операції над вмістом A1 і A2 записують за адресою A1. По аналогії з таким записом в мовах C, C++ з'явилась така синтаксична конструкція:

```
V ⊗ = exp; // замість V = V ⊗ exp;
```

Тут символ ⊗ позначає один із знаків бінарних операцій:

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

Наприклад:

```
x += 2;  //замість x=x+2;

z /= 1.5;     //замість z=z/1.5;
```

Досить екзотично виглядає оператор присвоєння, в правій частині якого через кому задана послідовність деяких дій. наприклад:

```
x=(y=sin(a+b),z=cos(a-b),max(y,z));
```

Дії в дужках виконуються зліва направо, тобто спочатку буде обчислене значення sin (a + b), яке запишеться в змінну ***y***, потім буде обчислене значення змінної z. Остаточним результатом дужок, який буде присвоєно змінній x, стане обчислення значення функції max (y, z).