Лекція 9-10

Операції порівняння та логічні операції в мові С. Порозрядні (побітові) операції. Унарні операції.

Операції порівняння застосовуються для порівняння (зіставлення) числової або символьної інформації. Результатом виконання операцій є або істина (*true*), або брехня (*false*). Перерахуємо ці операції:

Назва операції	Знак операції в С
Більше	>
Більше або дорівнює	>=
Менше	<
Менше або дорівнює	<=
Дорівнює	==
Не дорівнює	!=

Пріоритет цих операцій нижче, ніж у арифметичних операцій. Приклад використання: **`A+B>=C.`**

Перевіряємо: сума А і В більше С? Якщо «так», то відповіддю буде *true*, якщо "ні", то *false*.

У використанні операцій порівняння немає нічого складного. Але слід звернути увагу на одну обставину.

Дійсні числа в пам'яті зберігаються приблизно, при роботі з ними в ході виконання арифметичних операцій можуть накопичуватися похибки, тому не рекомендується виконувати перевірку на рівність або на нерівність для дійсних чисел, тому що результат може бути непередбачуваний.

Якщо для дійсних чисел треба виконати перевірку на рівність (типу A == B), то замінюємо її на перевірку на нерівність виду

| A-B | <= ε, де ε – деяка мала позитивна величина епсилон.

У термінах мови С ++ це буде виглядати так

fabs (A-B) <= eps

Приклади.

a>b

(a+b)<=2.5;

a>b==2 – значення виразу завжди буде мати значення *false*, так як a>b повертає результат або 0 або 1.

Логічні операції тісно пов'язані з операціями порівняння і використовуються для побудови складних логічних виразів. Є три логічні операції:

Назва	*Знак операції*	*Приклад запису*	*Пояснення*
Логічне заперечення (**НІ**)	**`!`**	**`!x`**	Якщо **`x`** — істина, то результат — не істина и навпаки

Логічне множення кон'юнкція (**І**)	**&&**	**X && Y**	Результат — істина, якщо істинні обидва операнда
Логічне додавання диз'юнкція (**АБО**)	**||**	**X || Y**	Результат — істина, якщо істиною є хоча б один операнд

Щоб продемонструвати виконання логічних операцій складемо програму, яка буде виводити таблицю істинності:

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
   bool a1 = true, a2 = false; // объявление логических переменных
   bool a3 = true, a4 = false;
   cout << "Tablica istinnosti log operacii &&" << endl;
   cout << "true  && false: " << ( a1 && a2 )   << endl // логическое И
        << "false && true: "  << ( a2 && a1 )   << endl
        << "true  && true: "  << ( a1 && a3 )   << endl
        << "false && false: " << ( a2 && a4 )   << endl;
   cout << "Tablica istinnosti log operacii ||" << endl;
   cout << "true  || false: " << ( a1 || a2 )   << endl // логическое ИЛИ
        << "false || true: "  << ( a2 || a1 )   << endl
        << "true  || true: "  << ( a1 || a3 )   << endl
        << "false || false: " << ( a2 || a4 )   << endl;
   cout << "Tablica istinnosti log operacii !"  << endl;
   cout << "!true: " << ( ! a1 ) << endl // логическое НЕ
        << "!false: "<< ( ! a2 ) << endl;
   system("pause");
   return 0;
}
```

При записі складних логічних виразів необхідно пам'ятати, що логічні операції мають нижчий пріоритет ніж арифметичні дії та операції порівняння крім операції заперечення, яка має найвищий пріоритет.

Як і в арифметичних виразах, можливе використання круглих дужок для зміни порядку обчислення операторів відношення або логічних операторів. наприклад:

! 1 && 0

дасть в результаті 0, оскільки ! обчислюється першим, а потім обчислюється &&. Якщо в цьому виразі поставити дужки наступним чином:

! (1 && 0) то вийде істина.

Приклади.

```
1)   x < y && y < z
2)   x == w || x == y || x == z
```

3) A>9 && A<100 – перевірка чи є число двозначним.

Порозрядні (побітові) операції

Як відомо, на комп'ютері в один прийом, тобто однією операцією, можна обробляти тільки дані розміром в 1 байт або більше. При цьому в ряді випадків необхідно з'ясувати значення окремих двійкових розрядів (бітів). Для цих цілей в мові С ++ є великий набір операцій, що дозволяють виконувати різні дії над окремими бітами. Все побітові операції допустимі тільки для цілих типів. Перерахуємо ці операції:

Назва	*Знак операції*	*Приклад запису*	*Результат виконання*
Поразрядне **І**	`&`	`3&5`	`1`
Поразрядне **АБО**	`|`	`3|5`	`7`
Поразрядне виключаюче **АБО**	`^`	`3^5`	`6`
Інверсія	`~`	`~3`	`-4`
Поразрядний зсув вліво	`<<`	`5<<1`	`10`
Поразрядний зсув вправо	`>>`	`5>>1`	`2`

При виконанні поразрядной операції ***І*** над відповідними розрядами вихідних операндів виконується операція І. Вона нагадує логічну операцію І, але тільки виконується з окремими бітами. Якщо відповідні біти в обох операндах рівні 1, то і відповідні біт результату дорівнює 1, інакше він дорівнює 0. Побітове виконання цієї операції наведено нижче:

	Двійкова форма			Десяткова
&	1	0	1	5
	0	1	1	3
	0	0	1	1

При виконанні поразрядной операції ***АБО*** все відбувається аналогічно, тільки над розрядами виконується операція АБО: результат дорівнює 0, якщо обидва відповідних розряду рівні 0, інакше результат дорівнює 1.

	Двійкова форма			Десяткова
|	1	0	1	5
	0	1	1	3
	1	1	1	7

Порозрядне ***виключає АБО*** дає нам інше правило для обчислення розрядів результату: результат дорівнює 1, якщо розряди різні, і дорівнює 0, якщо вони однакові.

	Двійкова форма			Десяткова
^	1	0	1	5
	0	1	1	3
	1	1	0	6

Інверсія – це одномісна операція, що замінює кожен біт операнда на зворотний: 0 на 1, а 1 на 0. Зверніть увагу: інверсія – це не зміна знака операнда.

	Двійкова форма			Десяткова
~	1	0	1	5
	0	1	0	2

Порозрядний зсув вліво виконується для розрядів лівого операнда на число позицій, рівне правому операнду. За результатом роботи зсув вліво на k позицій аналогічний множенню вихідного числа на 2^k.

змінна << число зсувів

Порозрядний зсув вправо виконується для розрядів лівого операнда на число позицій, рівне правому операнду. За результатом роботи зсув вправо на k позицій аналогічний цілочисленному поділу вихідного числа на 2^k.

змінна >> число зсувів

Операції зсуву працюють швидше, ніж операції множення або ділення.

Операції бітового зсуву можуть бути корисні при декодуванні інформації від зовнішніх пристроїв і для читання інформації про статус. Оператори бітового зсуву можуть також використовуватися для виконання швидкого множення і ділення цілих чисел. Зрушення вліво рівносильний множенню на 2, а зрушення вправо – поділу на 2, як показано в таблиці.

	Бітове представлення результату	значення x
char x;		
x = 7;	00000111	7
x = x << 1;	00001110	14
x = x << 3;	01110000	112
x = x << 2;	11000000	192
x = x >> 1;	01100000	96
x = x >> 2;	00011000	24

Бітові операції найбільш часто застосовуються при розробці драйверів пристроїв, наприклад програм для модемів, дисків і принтерів, оскільки бітові оператори можуть використовуватися для виключення деяких бітів, наприклад парності. (Біт парності використовується для підтвердження того, що інші біти в байті не змінювалися. Він, як правило, є старшим бітом в байті.).

Унарні операції &, *

Перш ніж розглядати зазначені операції згадаємо, як розташовуються дані у пам’яті. Кожному значенню відповідає наступний шаблон

адреса	ідентіфікатор	значення (згідно до типу)

Змінна – це область пам'яті, до якої ми звертаємося за даними, що знаходяться там, використовуючи ідентифікатор (в даному випадку, ім'я змінної). При цьому у цій поміченій ім'ям області є ще й адреса, яка виражається числом і зрозуміла комп'ютерній системі. Цю електронну адреса можна отримати і

записати в особливу змінну. Змінну, що містить адресу області пам'яті, називають покажчиком.

Покажчик – це адреса змінної в пам'яті. Покажчик на змінну – це змінна, спеціально створена для зберігання покажчика на об'єкт певного типу. Знаючи адресу змінної, можна істотно спростити роботу деяких програм. Покажчики мають три головних призначення в С:

- надають швидке звернення до елементів масиву;
- дозволяють функцій модифікувати передані параметри.;
- підтримують динамічні структури даних, наприклад списки.

Оператор **&**. Це унарний оператор, який повертає адресу операнда в пам'яті. Наприклад:

m = &count;

розміщує в *m* адресу змінної *count*. Це адреса внутрішнього розташування змінної в комп'ютері. З самим значенням змінної нічого не робиться. Оператор **&** можна запам'ятати як «взяття адреси». Тому вищезгаданий оператор присвоювання можна прочитати як «**m** отримує адресу *count*».

Для кращого розуміння даного присвоювання припустимо, що змінна *count* знаходиться за адресою 2000. Також припустимо, що *count* має значення 100. Після цього присвоювання *m* буде містити 2000.

Оператор *****, що доповнює **&**. Це унарний оператор, який повертає значення змінної за вказаною адресою. Наприклад: якщо *m* містить адресу змінної *count*, то

q = * m;

розміщує значення *count* в *q*. З вищенаведеного прикладу випливає, що *q* отримає значення 100, оскільки 100 зберігалося за адресою 2000 – адресою, що знаходиться в змінній *m*. Операцію ***** можна запам'ятати як «за адресою». В даному випадку оператор можна прочитати як «*q* отримує значення за адресою *m*».

Адреса 2000 m =&count =2000	Ідентифікатор count	значення (згідно до типу) 100 q = * m=100

На жаль, знаки для множення і для взяття «за адресою» – однакові, втім як і знаки бітового І та «взяття адреси». Ці оператори не мають зв'язку між собою. Як **&** так і ***** мають більш високий пріоритет у порівнянні з іншими арифметичними операціями, за винятком унарного мінуса, що має такий же пріоритет.

Змінні, що містять адреси або покажчики, повинні оголошуватися шляхом приміщення ***** перед ім'ям змінної для вказівки компілятору того, що змінна містить покажчик. Наприклад, для оголошення покажчика *ch* на символ, слід написати

char * ch;

Тут *ch* – це не символ, а покажчик на символ, в чому і полягає принципова відмінність. Тип даних, на який вказує покажчик, як в нашому випадку **char**, називається базовим типом покажчика. Сам покажчик – це змінна, яка використовується для зберігання адреси об'єкта базового типу. Отже, покажчик на

символ (або будь-який інший покажчик) має фіксований розмір, який визначається архітектурою комп'ютера, для зберігання адреси. Важливо запам'ятати, що покажчик слід використовувати тільки для вказівки на типи даних, що збігаються з базовим типом.

Можна змішувати оголошення покажчиків і звичайних змінних в одному рядку. наприклад:

int x, * y, count;

оголошує *x* і *count* як змінні цілочисельного типу, а *y* – як покажчик на цілочисельний тип.

Нижче оператори ***** і **&** використовуються для занесення числа 10 в змінну *target*:

```
#include <stdio.h>
/* присвоение с помощью * и & */
int main(void)
{
   int target, source;
  int *m;

source = 10;     m = &source;        target = *m;
printf("%d", target);
return 0; }
```

Додаткові можливості

Ansi драйвер

ANSI.SYS-драйвер для операційної системи MS-DOS,– розширення стандартного драйвера клавіатури і екрану CON, що входить до складу MS-DOS. Грунтується на керуючих послідовностях ANSI. Забезпечує додаткові функції управління дисплеєм: підтримку ANSI-графіки (завдання кольору символів і фону), позиціонування курсору, перепризначення клавіш і т. п

Завантажити цей драйвер можна за допомогою SETUP (або INSTALL) або відредагувавши безпосередньо файлу CONFIG.SYS і включивши в нього оператор: DEVICE = C:\NWDOS\ANSI.SYS

ESC-послідовність зазвичай має наступну форму: **Esc[**параметр

Нижче наведені деякі команди

Команда	**Последовательность**	**Действие**
CUP	ESC[y;xH	Позиціонування курсора в точку а заданими координатами. За замовчуванням курсор поміщається в лівий верхній кут екрану.
CUU	ESC[yA	Переміщує курсор вгору на *y* рядків.
CUD	ESC[yB	Переміщує курсор вниз на *y* рядків.
CUF	ESC[xC	Переміщує курсор вправо без зміни позиції в рядка
CUB	ESC[xD	Переміщує курсор вліво без зміни позиції в рядка.
SCP	ESC[s	Зберігає поточну позицію курсора.
RCP	ESC[u	Відновлює позицію курсора.

Для управління кольором використовують наступні команди

Команда	Колір	Команда	Колір
ESC[30m	чорний текст	ESC[31m	червоний текст
ESC[32m	зелений текст	ESC[33m	помаранчевий текст
ESC[34m	синій текст	ESC[35m	бузковий текст
ESC[36m	блакитний текст	ESC[37m	білий текст
ESC[40m	чорний фон	ESC[41m	червоний фон
ESC[42m	зелений фон	ESC[43m	помаранчевий фон
ESC[44m	Синій фон	ESC[45m	бузковиф фон
ESC[46m	блакитний фон	ESC[47m	білий фон

Приклад.

printf("\033[47m\033[31m"); – встановлення червоного тексту на білому фоні.

Esc-послідовності

Послідовності символів, що починаються зі зворотним косою риски, називаються керуючими **Esc**-послідовностями (escape-sequence).

Esc-послідовності використовуються:

- для запису символів, які не мають графічного зображення (Наприклад, \ a - звуковий сигнал);
- для запису символів: символу апострофа ('), зворотної косої риски (\), Знаку питання (?) і лапки (");
- для запису будь-якого символу за допомогою його шістнадцятирічного або вісімкового коду, наприклад, \ 073, \ 0xF5. Числове значення коду має перебувати в діапазоні від 0 до 255.

Таблиця 1.4 Допустимі ESC-послідовності в мові С ++

Представлення	Внутрішній код	Символ для позначення	Дія
'\a'	0x07	bel (audible bell)	звуковий сигнал
'\b'	0x08	bs (backspace)	повернення на крок
'\f '	0x0C	ff (form feed)	переведення сторінки
'\n'	0x0A	lf (line feed)	переведення рядка (LF)
'\r'	0x0D	cr (carriage return)	повернення каретки(CR)
'\t	0x09	ht (horizontal tab)	горизонтальна табуляція
'\v'	0x0B	vt (vertical tab)	вертикальна табуляція
'\\'	0x5C	\ (backslash)	зворотній слеш
'\' '	0x27	(single quote)	апостроф
'\"'	0x22	" (double quote)	лапки
'\?'	0x3F	? (question mart)	знак питання
'\000'	000	octal number	8-річний код
'\0xhh'	hh	hex number	16-річний код

Функція **beep(***dwFrec, dwDuration***)** – функція Windows API, призначена для відтворення простих однотонних звуків через вбудований динамік із заданою частотою і тривалістю. Існує в операційних системах сімейства Microsoft Windows

Функція Beep виконується синхронно, тобто повертає управління лише після завершення відтворення звуку. Вона має два параметри:

- *dwFrec* – частота звуку в герцах, допустимий діапазон - від 37 до 32 767 Гц;
- *dwDuration* – тривалість звуку в мілісекундах;

і повертає значення типу BOOL (нульове при успішному виконанні).

Мінімальне допустиме значення частоти звуку 37 Гц запобігає можливість випадкового або навмисного відтворення інфразвуку, який може становити небезпеку для здоров'я людини.

Прототип функції описаний в бібліотеці **windows.h**.

Приклад

```
#include <windows.h>

void beep(DWORD dwFreq, DWORD dwDuration) {
    Beep(dwFreq, dwDuration);
}

int main() {
    // Відтворити звук з частотою 1000 Гц протягом 500 мілісекунд
    beep(1000, 500);

    return 0;
      }
```