ОРГАНІЗАЦІЯ РОЗГАЛУЖЕНЬ. УМОВНИЙ ОПЕРАТОР ТАОПЕРАТОР ВИБОРУ

Блок команд

Частина команд, записаних всередині фігурних дужок { } – це блок команд.

```
{
  // команда 1;
  // команда 2;
  ...
  } // після блоку команд ; не ставимо
```

Блок може містити будь-які команди, включаючи команди визначення / оголошення даних та інші блоки (тоді ці блоки називаються вкладеними).

Всі імена, оголошені в блоці (якщо вони не оголошені як `static`), існують і видимі лише в цьому блоці. Після блоку команд крапку з комою не ставиться.

Реалізація розгалужень

Розгалуженням називається вибір програмою певної групи команд залежно від виконання певної умови, при цьому виконується лише одна з гілок алгоритму. Для програмної реалізації розгалужень використовуються оператори передавання керування, що дають змогу змінювати порядок виконання операторів програми.

У мові C++ для цього передбачено такі операції:

- безумовний перехід (`goto`);
- умовний перехід (`if`);
- вибір варіанта (`switch`).

Для запису умови переходу використовуються логічні (булеві) вирази.

Оператор безумовного переходу

Оператор безумовного переходу *goto* дає змогу передавати керування у будь-яке місце програми, позначене спеціальною міткою. Синтаксис

goto ім’я_мітки;

Мітку записують перед оператором, на який потрібно передати керування, і відокремлюють від нього символом двокрапки (:). Ім’я_мітки має бути визначене в цій ж функції (тобто, помічена цією міткою команда має бути записана в цій ж функції, в якій є команда goto). Вказана мітка – це «пункт призначення» для передачі управління поміченій команді. Наприклад

```
int test;
goto Go_;//оператор goto
test = 5;
test ++;
Go_: //мітка
test = 10;
```

Лише одна команда в певній функції може бути помічена певною міткою (не можна однією міткою помітити кілька команд, «пункт призначення» для команди переходу має бути лише один).

Хороший стиль – уникати команди goto. Замість команди goto слід використовувати команди break continue return if while do for switch.

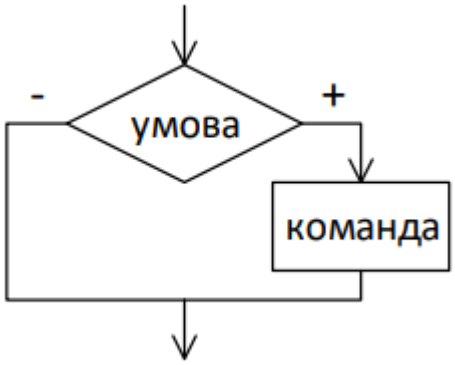
Оператор розгалуження if

Оператор розгалуження `if` має скорочену та повну форми. Якщо умова логічного виразу істинна (результат дорівнює *true* або має *ненульове значення*), то виконується оператор, записаний після умови, інакше виконується перехід на наступний оператор програми.

Оператор `if` має дві форми: скорочену та повну.

Скорочена форма оператора `if` має такий вигляд:

```
if (<умова>) <команда>;
```

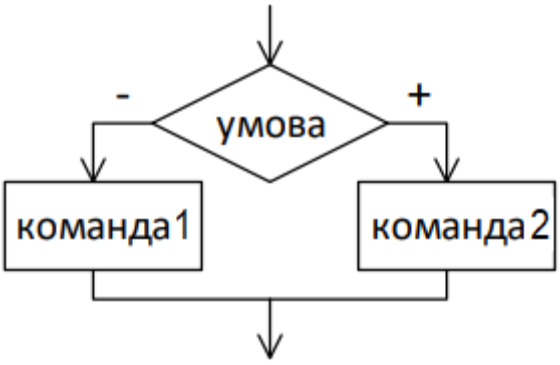


Повна форма

```
if (<умова>) <команда 1>;
    else <команда 2>;
```

Якщо ж у певній гілці потрібно виконати декілька операторів, то їх слід записати в операторних дужках {}.

```
if (<умова>)
{
<команда 1>;
<команда 2>;
}
else
{
<команда 3>;
<команда 4>;
}
```



Якщо умова логічного виразу істинна (значення логічного виразу ненульове), то виконується оператор1, інакше – виконується оператор2, після чого виконується перехід на наступний оператор.

Умову перевірки певної цілої змінної k на ненульове значення можна записати так `if`(k!=0) або if(k), оскільки ненульове значення сприймається як `true`. Тоді запис `if`(k==0) еквівалентний `if`(!k).

У конструкціях `if-else-if` умови операторів if перевіряються зверху донизу. Як тільки якась з умов набуває ненульового значення, одразу виконуватиметься оператор, який слідує за цією умовою, а останню частину конструкції буде проігноровано.

Типові помилки при організації розгалужень

•*Присвоєння замість порівняння на збіг*

= замість ==:

```
int x;
/* ... */
if (x = 1)
    cout << "x == 1" << endl;
else
    cout << "x != 1" << endl;
```

Присвоєння – це операція, результат якої = значенню, що було присвоєне. Наведений приклад завжди буде виводити x == 1, бо 1 трактується як `true`.

•*Ланцюжок порівнянь*

Припустимо, потрібно в якості умови розгалуження обчислити значення виразу 2 < x < 5. Якщо напишемо цю команду буквально:

```
if (2 < x < 5)
    cout << "x в діапазоні від 2 до 5" << endl;
else
    cout << "x поза діапазоном від 2 до 5" << endl;
```

і це приведе до помилки!

•*Вкладений `if` замість складної умови*

Не можна замість такої конструкції:

```
int x, y, z;
// ...
if (x > 0 && y > 0)
    z = 1;
else
    z = 2;
```

використовувати таку:

```
int x, y, z;
// ...
if (x > 0)
{
    if (y > 0)
        z = 1;
}
else
    z = 2;
```

Тернарна операція

У мові С є *тернарна* операція – це умовна операція, яка має такий формат: <умова> ? < вираз 1> : < вираз 2>;

При виконанні вираз_1 приводиться до типу `bool` та обчислюється його значення. Якщо вираз_1 має значення `true`, то обчислюється значення виразу_2, яке буде результатом усього умовного виразу. Якщо значення виразу_1 – `false`, то обчислюється значення виразу_3, яке стане результатом умовного виразу.

Наприклад

```
int i = 1, j = 2;
int k = i > j ? 3 : 4;
cout << k << endl;
```

еквівалентно такому фрагменту:

```
int i = 1, j = 2, k;
if (i > j) k = 3;
else k = 4;
cout << k << endl;
```

Нижче наведено приклад обчислення значення функції з використанням тернарного оператору

```
int x, z, y;
cout << "Please, input x=";
cin >> x;
(x <= -4) ? y = x * x - 8 : (x >= 0) ? y = 2 - x : y = 3 * x - 2;
cout << "\n" << y;
```

Оператор вибору варіантів

Команда вибору варіанта `switch` дозволяє вибрати один із багатьох варіантів розгалуження. Формат оператора вибору варіантів `switch` наступний:

`switch` (<вираз>)

```
{ case <значення_мітка_1> : <послідовність_операторів_1>;
 break;
 …
 case <значення_мітка_n> : <послідовність_операторів_N>;
 break;
 [ default: <послідовність_операторів>;
  break;]
}
```

Порядок обчислення в операторі `switch` такий: спочатку обчислюється вираз, записаний у дужках після слова `switch`. Цей вираз повинен бути цілого або символьного типу. Значення виразу порівнюється зі значеннями міток, що стоять після ключових слів `case`. Якщо значення виразу співпало зі значенням певної мітки, то виконується відповідна послідовність операторів, позначена цією міткою і записана після двокрапки, поки не зустрінеться оператор `break`.

Оператор break перериває виконання циклу або поточної альтернативи розгалуження чи вибору варіанта. Передає управління наступній команді після поточного структурного оператора. (використовується в командах циклів, if, switch). В даному прикладі оператор break здійснює вихід із switch. Якщо секція альтернатив case завершується командою break; то після виконання всіх команд цієї секції управління виходить за межі блоку switch. Якщо наприкінці операторів певної гілки case відсутній оператор break, то будуть почергово виконані всі оператори до наступного `break` чи до кінця `switch` для всіх гілок `case` незалежно від значення їхніх міток.

Якщо не спрацювала жодна перевірка жодної секції альтернатив `case`, то управління передається в секцію `default` (якщо вона є). Якщо не спрацювала жодна перевірка жодної секції альтернатив `case`, і немає секції `default`, то управління виходить за межі блоку `switch`.Наприклад, наступний фрагмент дозволяє за введеним номером дня тижня вивести його назву.

```
int num;
cout << "Input number from 1 to 7.\nnum =";
cin >> num;
switch (num)
{
case 1: cout << "mondey"; break;
case 2: cout << "tuesday"; break;
case 3: cout << "wednesday"; break;
case 4: cout << "thursday"; break;
case 5: cout << "friday"; break;
case 6: cout << "saturday"; break;
case 7: cout << "sunday"; break;
default: cout << "Error!"; break;
}
```

В наступному прикладі для двох значень змінної перебдачена єдина дія:

```
#include <iostream>
#include <iomanip>
using namespace std;

 int main()
{
	int x;
	switch (x)
	{
	case 1:
	case 2:
	  cout << "x = 1 or x = 2" << endl;
	  break;
	case 3:
	  cout << "x = 3" << endl;
	case 4:
	  cout << "x = 4" << endl;
	  break;
	default:
	  cout << "x has another value" << endl;
	}
	return 0;
}
```