**Лекція 3**

**Оператори умови**

**План**

1. Оператори
2. Оператор if
3. Вкладені оператори if
4. Тернарна альтернатива
5. Оператор switch
6. Вкладені оператори switch

*Оnepamop —* це частина програми, яку можна виконати окремо. Іншими словами, оператор визначає якусь дію. Оператори мови С и C++ розділяються на наступні категорії.

• Умовні оператори

• Оператори циклу

• Оператори переходу

• Мітки

• Оператора-виразів

• Блоки

До *умовного* (conditional) ставляться оператори **if** й **switch.** (Умовні оператори іноді називають *операторами розгалуження* (selection statement).) *Оператори циклу* (iteration statements) позначаються ключовими словами while, **for** й **do while.** Група операторів переходу складається з операторів **break, continue, goto** й **return.** Мітками служать оператори **case, default** (вони розглядаються в розділі "Оператор switch") і властиво мітки, які описуються в розділі "Оператор goto". Оператори-вираження — це оператори, що складаються із припустимих виражень. Блок являє собою фрагмент тексту програми, ув'язнений у фігурні дужки. Іноді блоки називають *складеними операторами* (compound statements).

**Оскільки** в багатьох операторах результат обчислення залежить від істинності або хибності деяких перевірок, почнемо з понять "істина" й "неправда".

**Істині та хибні значення в мовах С и C++**

При виконанні багатьох операторів мови C/C++ обчислюються значення умовних виразів, які мають істині або хибні значення. У мові С++ істинним уважається будь-яке ненульове значення, у тому числі негативне. Хибне значення завжди дорівнює нулю. Таке подання істинних або хибних значень дозволяє створювати надзвичайно ефективні програми.

Поряд із цим у мові C++ використовується булевий тип даних з ім'ям **bool,** що передбачає тільки два значення: **true й false.** У мові C++ число 0 автоматично перетворює в значення **false,** а будь-яке ненульове значення — у значення **true.** Справедливо й зворотне твердження: значення **false** перетвориться в 0, a **true** — в **1. З** формальної точки зору умовне вираження, що входить в умовний оператор, має тип **bool.** Однак, оскільки будь-яке ненульове значення перетвориться в значення **true,** а число 0 — у значення **false,** між мовами С и C++ щодо цього немає ніякої різниці.

**Оператор if**

Оператор if має такий вигляд:

іf (*вираз) оператор;* [else *оператор;*]

Тут *оператор* може складатися з одного або декількох операторів або бути відсутнім зовсім (порожній оператор). Розділ else є необов'язковим.

Якщо *вираз* істинний (тобто не дорівнює нулю), виконується оператор або блок, зазначений у розділі if, у противному випадку виконується оператор або блок, передбачений у розділі else. Оператори, зазначені в розділах if або else, є взаємовиключними.

У мові С результатом умовного виразу є *скаляр,* тобто ціле число, символ, покажчик або число із плаваючою крапкою. У мові C++ до цього набору типів додається тип bool. Число із плаваючою крапкою рідко застосовується як результат умовного вираження, що входить в умовний оператор, оскільки значно сповільнює виконання програми. (Це пояснюється тим, що операції над числами із плаваючою крапкою виконуються повільніше, ніж над цілими числами або символами.)

**Вкладені оператори if**

*Вкладеним* (nested) називається оператор **if,** що перебуває усередині іншого оператора **if** або **else.** Вкладені оператори **if** зустрічаються досить часто. У вкладеному умовному операторі розділ **else** завжди пов'язаний з найближчим оператором **if,** щоперебуває з ним в одному блоці й не пов'язаним з іншим оператором **else.** Розглянемо приклад.

if (i)

**{**

if(j) оператор1;

if(k) оператор2; /\* даний if \*/

else операторз; /\* пов'язаний з даним оператором else \*/

**}**

else оператор 4; /\* пов'язаний з оператором if(i) \*/

Останній розділ **else** зв'язаний не з оператором **if** (j), що перебуває в іншому блоці, а з оператором **if(i).** Внутрішній розділ **else** пов'язаний з оператором **if** (к), тому що цей оператор **if** є найближчим.

Набагато важливіше, що мова C++ допускає до 256 рівнів вкладення. Однак на практиці глибоко вкладені умовні оператори використаються вкрай рідко, оскільки це значно ускладнює логіку програми.

**Тернарна альтернатива**

Замість операторів **if-else** можна використати тернарний оператор "?". Загальний вид оператора **if-else** виглядає в такий спосіб.

**if** *(умова) вираз;* **else** *вираз;*

Однак у цьому випадку з операторами **if** й **else** зв'язані окремі вирази, а не оператори.

Оператор "?" називається *тернарним,* оскільки має три операнда. Його загальний вид такий.

*Вираз1 ? Вираз2: Вираз3* Зверніть увагу на використання й місце розташування двокрапки.

Оператор "?" виконується в такий спосіб. Спочатку обчислюється *Вираз 1.* Якщо він є істинним, обчислюється *Вираз 2,* і його значення стає значенням усього тернарного оператора. Якщо *Вираз 1* є хибним, обчислюється *Вираз 3,* і результатом виконання тернарного оператора вважається саме його значення. Розглянемо приклад.

х = 10;

у = х>9 ? 100 : 200;

У цьому випадку змінній ***у*** привласнюється значення 100. Якби змінна ***х***була менше 9, то змінна в одержала б значення 200. Цей код можна переписати за допомогою операторів **if-else.**

х = 10;

if(x>9) у = 100;

else у = 200;

Тернарний оператор можна використати замість конструкції **if-else** не тільки для присвоєння значень. Як відомо, всі функції повертають яке-небудь значення (крім функцій, що повертають значення типу void). Отже, замість виразів в операторі "?" можна використати виклики функцій. Якщо в операторі "?" зустрічається ім'я функції, вона викликається, а її результат використається замість значення відповідного виразу. Це означає, що, використовуючи виклики функцій у якості операндів тернарного оператора, можна виконати одну або кілька функцій відразу. Розглянемо приклад.

#include <stdio.h>

int f1(int n);

int f 2(void);

int main(void)

{

int t;

cout<<"Уведіть число: "; cin>>t;

/\* Вивід відповідного повідомлення \*/

t ? fl(t) + f2() : cout<<"Уведений нуль.\n";

return 0;

}

int fl(int n)

{

cout<<n;

return 0;

}

int f2(void)

{

cout<<”уведено " ;

return 0;

}

**Оператор switch**

У мові C/C++ передбачений оператор різноманітного розгалуження **switch,** що послідовно порівнює значення виразів зі списком цілих чисел або символьних констант. Якщо виявляється збіг, виконується оператор, пов'язаний з відповідною константою. Оператор **switch** має такий вигляд.

**switch** *(вираз)*

**{**

**case** *констант1:*

*послідовність операторів*

**break;**

**case** *констант2:*

*послідовність операторів*

**break;**

**case** *константа3:*

*послідовність операторів*

**break;**

**.**

**.**

**.**

**default:**

*послідовність операторів*

**}**

Значенням *виразу* повинне бути символ або ціле число. Наприклад, вираз, результатом яких є число із плаваючою крапкою, не допускаються. Значення виразу послідовно порівнюється з константами, зазначеними в операторах **case.** Якщо виявляється збіг, виконується послідовність операторів, пов'язаних з даним оператором **case,** поки не зустрінеться оператор **break** або не буде досягнутий кінець оператора **switch.** Якщо значення виразу не збігається з жодною з констант, виконується оператор **default.** Цей розділ оператора **switch** є необов'язковим. Якщо він не передбачений, під час відсутності збігів не буде виконаний жоден оператор.

Стандарт мови C++ передбачає до 16384 операторів **case!** На практиці кількість розділів **case** в операторі **switch** варто обмежувати, оскільки воно впливає на ефективність програми. Незважаючи на те що оператор **case** є міткою, він використається тільки усередині оператора **switch.**

Оператор **break** ставиться до групи операторів переходу. Його можна використати як в операторі **switch,** так й у циклах. Коли потік керування досягає оператора **break,** програма виконує перехід до оператора, що знаходиться за оператором **switch.**

Варто знати три важливих властивості оператора **switch.**

• Оператор **switch** відрізняється від оператора if тим, що значення його виразу рівняється винятково з константами, у той час як в операторі if можна виконувати які завгодно порівняння або обчислювати будь-які логічні вираження.

• Дві константи в різних розділах **case** не можуть мати однакових значень, за винятком, коли один оператор **switch** вкладений в іншій.

• Якщо в операторі **switch** використаються символьні константи, вони автоматично перетворюють у цілочисельні.

З формальної точки зору наявність оператора **break** усередині оператора **switch** не обов'язково. Ці оператори перериває виконання послідовності операторів, пов'язаних з відповідною константою. Якщо його пропустити, будуть виконані всі наступні оператори **case,** поки не зустрінеться наступний оператор **break,** або не буде досягнутий кінець оператора **switch.** Наприклад, наведена нижче функція використає цей ефект для обробки інформації, що надходить на вхід драйвера.

/\* Обробка значення \*/

void inp\_handler(int i)

{

int flag;

flag = -1;

switch(i) {

case 1: /\* Ці оператори case мають загальну \*/

case 2: /\* послідовність операторів. \*/

case 3:

flag = 0;

break;

case 4:

flag = 1;

case 5:

error(flag);

break;

default:

process(i);

}

}

Цей приклад ілюструє дві властивості оператора **switch**. По-перше, оператор **case** може не мати пов'язаної з ним послідовності операторів. У цьому випадку потік керування просто переходить до наступного оператора **case**, як би "провалюючись" униз. У нашому прикладі три перших оператори **case** пов'язані з однієї й тією же послідовністю операторів, а саме:

flag = 0;

break;

По-друге, якщо оператор **break** відсутній, виконується послідовність операторів, зв'язана з наступним оператором **case**. Якщо значення i дорівнює 4, змінній **flag** привласнюється число 1, і, оскільки оператора **break** наприкінці даного розділу **case** відсутній, виконання оператора **switch** триває, і викликається функція **error (flag)**. Якщо значення i дорівнює 5, функція **error** буде викликана з параметром **flag**, рівним -1, а не 1.

Те, що під час відсутності оператора **break** оператори **case** виконуються один за іншим, дозволяє уникнути непотрібного дублювання операторів і підвищити ефективність програми.

**Вкладені оператори switch**

Оператори switch можуть бути вкладені. Навіть якщо константи розділів **case** зовнішнього й внутрішнього операторів **switch** збігаються, проблеми не виникають. Наприклад, наведений нижче фрагмент програми є цілком прийнятним.

switch(х) {

case 1:

switch(у) {

case 0: printf("Ділення на нуль.\п");

break;

case 1: process(x,у);

}

break;

case 2:

.

.

**Оперетор умовного переходу if-else**

#include <iostream.h>

void main()

{

float a,x,y;

cin>>x>>a;

if (x>2&&x<3) y=x\*a;//якщо x>2 й x<3 те y=x\*a

else if (x>=3){a=3;y=x+a;}//інакше, якщо x>=3 те a=3;y=x+a

else y=a;//інакше y=a

cout<<y;

}

**Оператор switch**

#include <iostream.h>

void main()

{

int x;

float y;

cin>>x;

switch (x)

{

case 1:y=x;break;//якщо x=1 те y=x

case 2:y=x\*x;break;//якщо x=2 те y=x\*x

case 3:y=x\*x\*x;break;//якщо x=3 те y=x\*x\*x

default: y=0;//в інших випадках y=0

}

cout<<y;

}

**Тернарний оператор ?:**

#include <iostream.h>

void main()

{

float x,y;

cin>>x;

y=(x>2||x==0)?x\*x:x\*x+2;//якщо x>2 або x==0 те y=x\*x, інакше y=x\*x+2

cout<<y;

}

**Використана література**

1. Г. Шилдт, Полный справочник по С++, 4-е видання, в-во «Вильямс», 2006
2. Х.М.Дейтел, Как програмировать на С++, 4-е видання, в-во «Бином-Пресс» 2009.
3. Р. Лафоре, Объектно-ориентированное программирование в С++, в-во «Питер», 2004, с 924.