

## **Занятие №2. Конструкции принятия решений**

*Конструкции принятия решений* позволяют приложениям проверять условия и выбирать направление действий.

Другим аспектом конструкций принятия решений в C++ являются *циклы*, которые выполняют повторяющиеся задачи. Они позволяют компьютеру повторять некоторые действия, не обременяя программиста необходимостью вновь и вновь вводить небольшие изменения в одни и те же места программы при каждом их повторном выполнении.

## Одноальтернативный оператор if

Общий синтаксис одноальтернативного оператора if имеет вид

```
if (условие) оператор;
```

для единственного исполняемого оператора, и

```
if (условие)  
{  
<последовательность операторов>  
}
```

для последовательности исполняемых операторов.

## Примеры

```
if (numberOfLines < 0)
    numberOfLines = 0;
```

```
if (numberOfLines < 0) numberOfLines = 0;
```

```
if ((height - 54) < 3)
{
    area = length * width;
    volume = area * height;
}
```

```
if ((height - 54) < 3) { area = length * width; volume = area * height;}
```

```
1: // Программа, демонстрирующая одноальтернативный
2: // оператор if
3: #include <iostream>
4: using namespace std;
5: int main ()
6: {
7:     double x;
8:     cout << "Enter a non-zero number: ";
9:     cin >> x;
10:    if ( x != 0)
11:        cout << "The reciprocal of " << x
12:        << " is " << (1/x) << endl;
13:    return 0;
14: }
```

Вот пример сеанса работы программы, представленной в листинге

```
Enter a non-zero number: 25
The reciprocal of 25 is 0.04
```

## **Двухальтернативный оператор if-else**

В двухальтернативной форме оператора if ключевое слово else отделяет друг от друга операторы, которые используются при выполнении каждой из альтернатив.

Двухальтернативный оператор if-else обеспечивает два альтернативных направления действий в зависимости от значения проверяемого булева условия.

Общий синтаксис двухальтернативного оператора if-else имеет вид:

```
if (условие)
    оператор1;
else
    оператор2;
```

для единственного исполняемого оператора в каждом из предложений,

**и**

```
if (условие)
{
    <последовательность операторов #1>
}
else
{
    <последовательность операторов #2>
}
```

для последовательности исполняемых операторов в обоих предложениях.

## Пример

```
if (moneyInAccount > withdraw)
{
moneyInAccount = withdraw;
cout << "You withdraw $" << withdraw << endl;
cout << "Balance is $" << moneyInAccount <<
endl;
}
else
{
cout << "Cannot withdraw $" << withdraw <<
endl;
cout << "Account has $" << moneyInAccount <<
endl;
}
```



```
1: // Программа, демонстрирующая двухальтернативный
2: // оператор if
3: #include <iostream>
4: #include <ctype.h>
5: using namespace std;
6: int main ()
7: {
8:     char c;
9:     cout << "Enter a letter: ";
10:    cin >> c;
11:    // преобразовать в прописную букву
12:    c = toupper (c);
13:    if (c >= 'A' && c <= 'Z')
14:    cout << "You entered a letter" << endl;
15:    else
16:    cout << "Your input was not a letter" << endl;
17:    return 0;
18: }
```

**Вот пример сеанса работы программы, представленной в листинге :**

```
Enter a letter: g
You entered a letter
```

## Многоальтернативный оператор if-else

*Многоальтернативный* оператор if-else содержит вложенные операторы if-else.

Многоальтернативный оператор if-else имеет следующий общий синтаксис

```
if (проверяемое_условие1)
оператор1; |{ < последовательность операторов #1 > }|
else if (проверяемое_условие2)
оператор2; |{ < последовательность операторов #2 > }|
else if (проверяемое_условиеN)
операторN; |{ < последовательность операторов #N > }
[else
операторN+1; |{ последовательность операторов #N+1 > }]
```

## Пример

```
char op;  
int opOk = 1;  
double x, y, z;  
cout << "Enter operand1 operator operand2: ";  
cin >> x >> op >> y;  
if (op == '+')  
    z = x + y;  
else if (op == '-')  
    z = x - y;  
else if (op == '*')  
    z = x * y;  
else if (op == '/' && y != 0)  
    z = x / y;  
else  
    opOk = 0;
```

Многоальтернативный оператор if-else выполняет ряд последовательных проверок до тех пор, пока не произойдет одно из следующих событий:

1. Одно из условий в предложении if или в предложении else if имеет значение true. В этом случае выполняются соответствующие операторы.
2. Ни одно из проверяемых условий не имеет значения true. Программа выполняет операторы во всеохватывающем предложении else (если оно существует).

```
1: // Программа, демонстрирующая многоальтернативный
2: // оператор if
3: #include <iostream>
4: using namespace std;
5: int main ()
6: {
7: char c;
8: cout << "Enter a character: ";
9: cin >> c;
10: if (c >= 'A' && c <= 'Z')
11:     cout << "You entered an uppercase letter" << endl;
12: else if (c >= 'a' && c <= 'z')
13:     cout << "You entered an lowercase letter" << endl;
14: else if (c >= '0' && c <= '9')
15:     cout << "You entered a digit" << endl;
16: else
17:     cout<<"You entered a non-alphanumeric character"<<endl;
18: return 0;
19: }
```

**Вот пример сеанса работы программы, представленной в листинге:**

```
Enter a character:  !
You entered a non-alphanumeric character
```

## Оператор switch

Оператор switch предлагает специальную форму создания многоальтернативного решения. Это позволяет вам исследовать разнообразные значения выражения (тип которого совместим с целым) и выбирать соответствующее направление действий.

Общий синтаксис оператора switch имеет вид:

```
switch (выражение)
```

```
{
```

```
  case constant1_1:
```

```
  [case constant1_2: ...]
```

```
  <один или несколько операторов>
```

```
  break;
```

```
  case constant2_1:
```

```
  [case constant2_2: ...]
```

```
  <один или несколько операторов>
```

```
  break;
```

```
  case constantN_1:
```

```
  [case constantN_2: ...]
```

```
  <один или несколько операторов>
```

```
  break;
```

```
  default:
```

```
  <один или несколько операторов>
```

```
  break;
```

```
}
```

## Пример

```
ok = true;
switch (op)
{
    case '+':
        z = x + y;
        break;
    case '-':
        z = x - y;
        break;
    case '*':
        z = x * y;
        break;
    case '/':
        if (y != 0) z = x / y;
        else
            ok = false;
        break;
    default:
        ok = false;
        break;
}
```



## Правила использования оператора switch

1. Switch требует совместимого с целым значения. Это значение может быть константой, переменной, вызовом функции или выражением. Оператор switch не работает с типами данных с плавающей точкой.
2. Значение после каждой метки case *должно* быть константой.
3. C++ не поддерживает метки case с диапазоном значений. В этом случае каждое значение из диапазона должно появляться с отдельной меткой case.

## Правила использования оператора `switch`

4. Окончание оператора `case` обычно отмечается словом `break`. Это вызывает переход к выполнению первого оператора, который следует после `switch`. Если вы не включаете `break`, то выполнение будет продолжаться со следующего оператора `case`.

Как альтернативу вместо `break` можно использовать оператор `return`. Это вызовет завершение работы текущей функции, если текущая функция — `main`, то программа завершится.

5. Предложение `default` — всеохватывающее, но оно не обязательно, если вы хотите проверить только отдельный ряд случаев.

6. Ряд операторов в каждой метке `case` или в групповых метках `case` можно не заключать в фигурные скобки.

# **Вложенные конструкции принятия решений**

Конструкции внешнего уровня помогают вам проверять предварительные или более общие условия. Внутренние конструкции помогают вам иметь дело с более специфическими условиями.