Лекция: Условные и циклические операторы в C++

Условные и циклические операторы — это базовые инструменты для управления потоком выполнения программы. Они позволяют изменять поведение программы в зависимости от условий и повторять выполнение участков кода несколько раз.

---

1. Условные операторы

Условные операторы используются для выполнения определённых действий при выполнении или невыполнении определённых условий.

a) Оператор `if`

Оператор `if` позволяет выполнить блок кода, если условие истинно (равно `true`). Если условие ложно (`false`), этот блок кода будет пропущен.

- **Синтаксис:**

```
if (условие) {
    // код выполняется, если условие истинно
}
```

- **Пример:**

```
int a = 5;
if (a > 0) {
    cout << "a больше 0";
}
```

b) Оператор `if-else`

Оператор `if-else` добавляет блок `else`, который выполняется, если условие в `if` ложно.

- **Синтаксис:**

```
if (условие) {
    // код выполняется, если условие истинно
} else {
    // код выполняется, если условие ложно
}
```

- **Пример:**

```
int a = -5;
if (a > 0) {
    cout << "a больше 0";
} else {
    cout << "a меньше или равно 0";
}
```

c) Оператор `else if`

Если нужно проверить несколько условий, используется конструкция `else if`. Она позволяет выполнить разные блоки кода в зависимости от множества условий.

- **Синтаксис:**

```
if (условие1) {
    // выполняется, если условие1 истинно
} else if (условие2) {
    // выполняется, если условие1 ложно, а условие2 истинно
} else {
    // выполняется, если все условия ложны
}
```

- **Пример:**

```
int a = 0;
if (a > 0) {
    cout << "a больше 0";
} else if (a == 0) {
    cout << "a равно 0";
} else {
    cout << "a меньше 0";
}
```

d) Тернарный оператор `? :`

Тернарный оператор — это краткая форма записи условного выражения, когда одно из двух значений выбирается в зависимости от условия.

- **Синтаксис:**

```
условие ? выражение1 : выражение2;
```

- **Пример:**

```
int a = 10;
string result = (a > 0) ? "Положительное число" : "Отрицательное число";
cout << result;
```

---

2. Оператор `switch`

Оператор `switch` используется для выбора одной из нескольких возможных ветвей выполнения программы на основе значения переменной.

a) Синтаксис оператора `switch`:

```
switch (выражение) {
    case значение1:
        // код выполняется, если выражение равно значению1
        break;
    case значение2:
        // код выполняется, если выражение равно значению2
        break;
    // другие варианты
    default:
        // код выполняется, если ни одно из значений не совпало
}
```

- **Пример:**

```
int day = 3;
switch (day) {
    case 1:
        cout << "Понедельник";
        break;
    case 2:
        cout << "Вторник";
        break;
    case 3:
        cout << "Среда";
        break;
    default:
        cout << "Неизвестный день";
}
```

- `break` используется для выхода из `switch` после выполнения одного из случаев. Если не использовать `break`, выполнение перейдёт к следующему случаю.

b) Оператор `switch` vs. `if-else`

- `if-else` более универсален, так как может проверять любые условия (больше, меньше, равно и т.д.).
- `switch` удобен для выбора между фиксированными значениями одной переменной (например, целые числа или символы).

---

3. Циклы

Циклы позволяют повторять выполнение блока кода до тех пор, пока выполняется определённое условие.

a) Цикл `while`

Цикл `while` выполняет блок кода, пока условие истинно. Условие проверяется перед каждой итерацией.

- **Синтаксис:**

```
while (условие) {
    // код выполняется, пока условие истинно
}
```

- **Пример:**

```
int i = 0;
while (i < 5) {
    cout << i << " ";
    i++;
}
```

b) Цикл `do-while`

Цикл `do-while` похож на `while`, но его отличительная особенность в том, что он выполняет код хотя бы один раз, так как условие проверяется после выполнения блока.

- **Синтаксис:**

```
do {
    // код выполняется как минимум один раз
} while (условие);
```

- **Пример:**

```
int i = 0;
do {
    cout << i << " ";
    i++;
} while (i < 5);
```

c) Цикл `for`

Цикл `for` используется, когда известно, сколько раз нужно выполнить блок кода. Он включает в себя три выражения: инициализация, условие, шаг.

- **Синтаксис:**

```
for (инициализация; условие; шаг) {
    // код выполняется, пока условие истинно
}
```

- **Пример:**

```
for (int i = 0; i < 5; i++) {
    cout << i << " ";
}
```

d) Цикл `for` с диапазоном (range-based for)

Этот вид цикла используется для итерации по элементам коллекций, таких как массивы или векторы.

- **Синтаксис:**

```
for (тип_элемента переменная : коллекция) {
    // код выполняется для каждого элемента коллекции
}
```

- **Пример:**

```
int arr[] = {1, 2, 3, 4, 5};
for (int x : arr) {
    cout << x << " ";
}
```

---

4. Операторы управления циклом

Иногда в процессе выполнения цикла требуется прервать его выполнение или пропустить определённые итерации.

a) Оператор `break`

Оператор `break` используется для немедленного завершения выполнения цикла. Обычно применяется для выхода из цикла при выполнении определённого условия.

- **Пример:**

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break;  // прерывает цикл, когда i равно 5
    }
    cout << i << " ";
}
```

b) Оператор `continue`

Оператор `continue` пропускает текущую итерацию цикла и переходит к следующей.

- **Пример:**

```
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        continue;  // пропускает текущую итерацию для чётных чисел
    }
    cout << i << " ";
}
```

c) Оператор `goto`

Оператор `goto` используется для перехода к метке в коде. Использование `goto` считается плохой практикой, так как оно делает код сложным для понимания и сопровождения.

- **Синтаксис:**

```
goto метка;

метка:
// код
```

- **Пример:**

```
int i = 0;
start:
cout << i << " ";
i++;
if (i < 5) {
    goto start;
}
```

Заключение

Условные операторы (`if`, `else`, `switch`) позволяют управлять выполнением программы в зависимости от условий, а циклы (`for`, `while`, `do-while`) обеспечивают многократное выполнение блоков кода. Управление выполнением с помощью операторов `break`, `continue` и других помогает эффективно работать с циклами и условными конструкциями.