Основы языков C/C++

Кафедра ЭО

Петрухин О.М.

Логический тип данных

Логический тип данных может принимать одно из двух значений:

- true (1)
- false (0)

Все сравнения в языке приводятся именно к этому типу

```
bool t = true, f = false;
std::cout << t << std::endl;
std::cout << f << std::endl;
```

Операторы сравнения

Операция проверки на равенство и отношение	Выражение	Результат
Равенство	x == y	**Равенство** возвращает **true**, если операнды равны
	x != y	**Неравенство** возвращает **true**, если операнды не равны
Отношение	x > y	**Больше чем** возвращает true, если левый операнд больше правого
	x < y	**Меньше чем** возвращает **true**, если левый операнд меньше правого
	x >= y	**Больше или равно** возвращает **true**, если левый операнд больше или равен правому
	x <= y	**Меньше или равно** возвращает **true**, если левый операнд меньше или равен правому

Оператор равенства

```
bool t = true, f = false;
bool res;
res = (t == f); // false
res = (t == t); // true
res = (f == f); // true
res = (1 == 1); // true
res = (1 == 2); // false
res = (f == (1 == 2)); //
???
```

```
int i1, i2;
bool res;
i1 = 1 + 2 + 3;
i2 = 6;
res = (i1 == i2); // true
char i1, i2;
i1 = 'a';
i2 = 'A';
res = (i1 == i2); // false
```

Операторы неравенства и сравнениия

```
bool t = true, f = false;
bool res;
res = (t != f); // true
res = (1 != 2); // true
res = (t != (1 != 2)); // ???
i1 = 8 - 3;
i2 = 6;
res = (i1 != i2); // true
```

Оператор «меньше чем»:

```
res = ( 2 < 3); // true
res = ( 2 < 2); // false
```

Оператор «меньше или равно»:

```
res = ( 2 <= 3); // true
res = ( 2 <= 2); // true
```

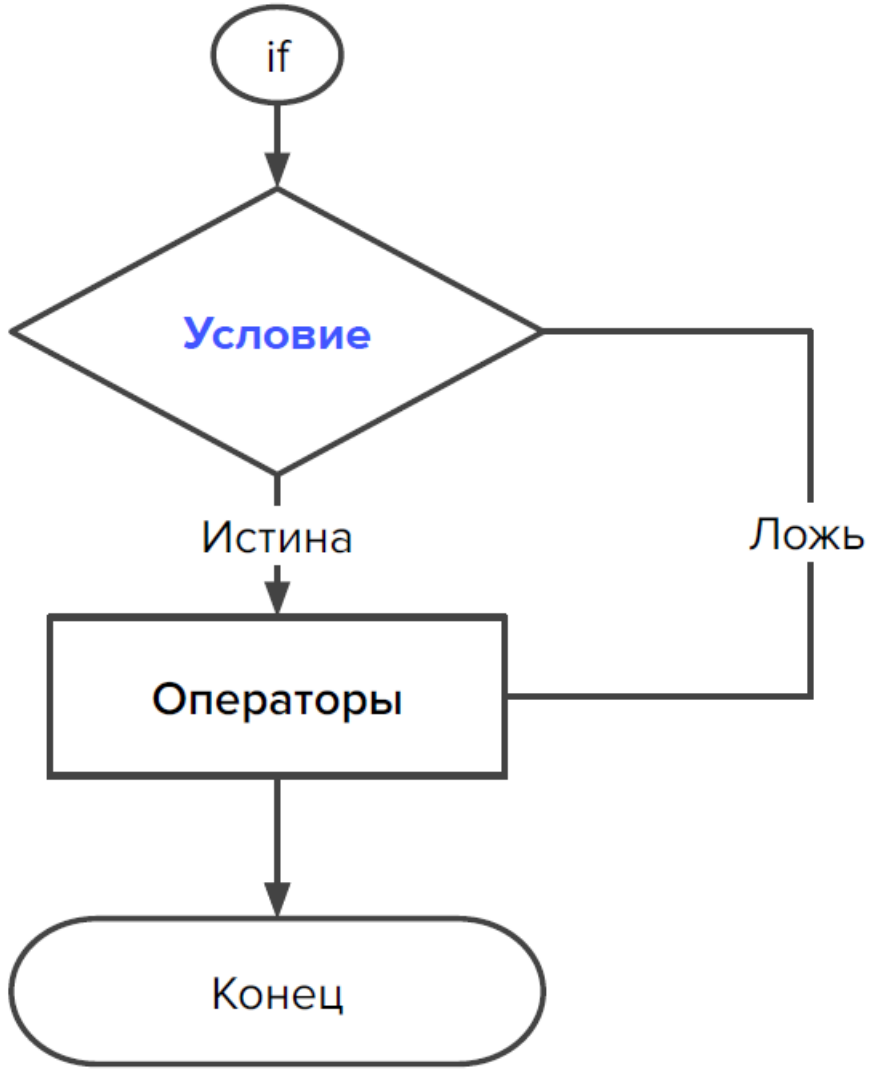
Операторы ветвления

Оператор if

Если условие истинно, то будут выполнены операторы в блоке **if**, иначе программа продолжит работу без выполнения операторов

```
if (условие){
    // Сделать одно;
    // Сделать второе;
    // Сделать третье;
}

if (условие)
    // Сделать что-то;
```



Операторы ветвления

Оператор if-else

Конструкция **if-else** позволяет выполнять два блока кода: для истинного и ложного условий. Это позволяет управлять поведением программы в зависимости от входных данных и состояний.

```
if (условие) {
    // Сделать одно;
    // Сделать второе;
} else {
    // Сделать другое;
    // И что-то ещё;
}
```

Операторы ветвления

Оператор if-else

В операторе **if-else**, если условное выражение имеет значение **true**, выполняется первый блок. Если условное выражение имеет значение **false**, выполняется второй блок. Так как условное выражение не может одновременно иметь значения **true** и **false**, блоки **if-else** не могут выполняться одновременно

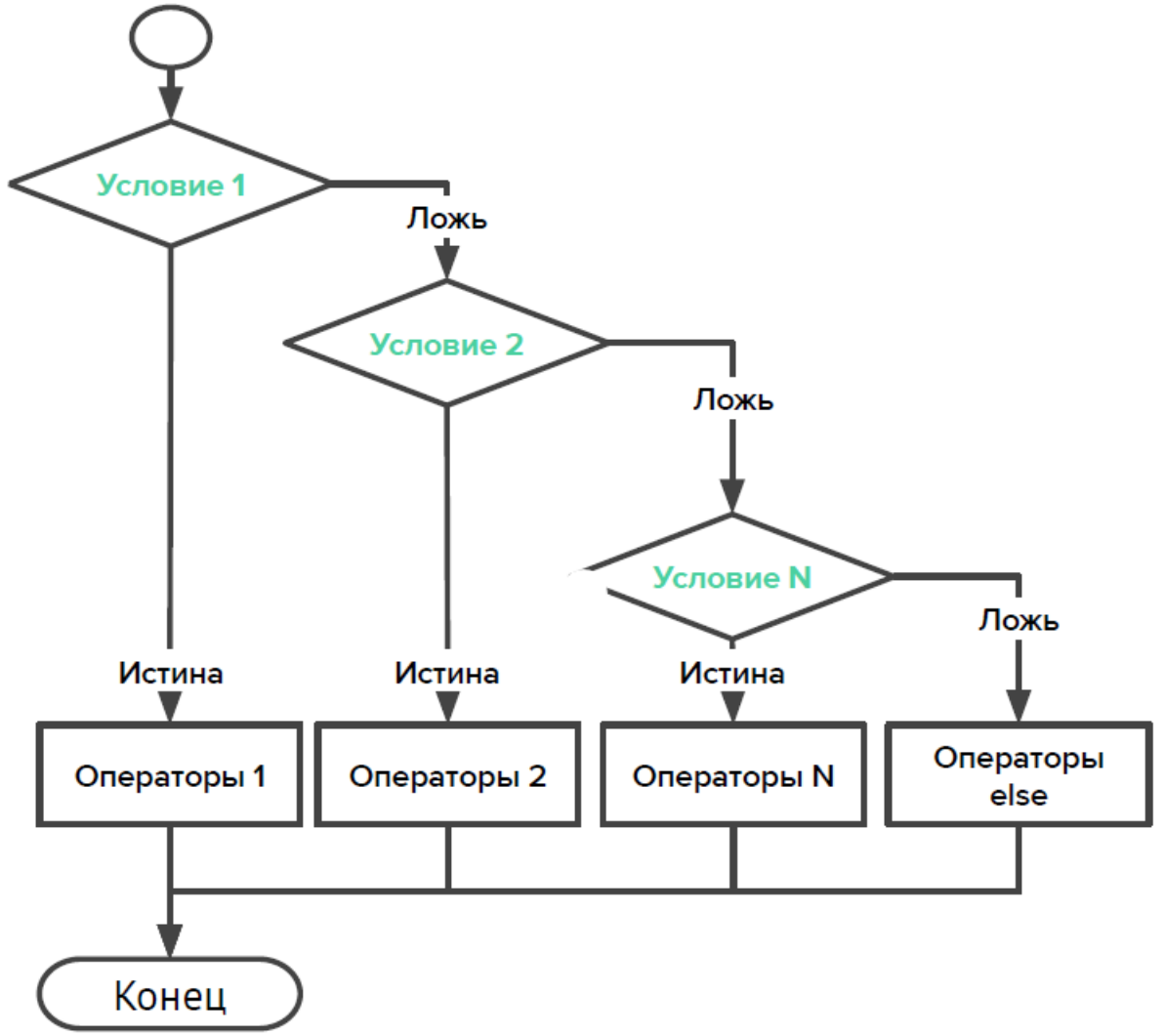
```
bool a = true;
if (a) {
    std::cout << "a is true" << std::endl;
} else {
    std::cout << "a is false" << std::endl;
}
```

Операторы ветвления

Оператор if-else-if-else

Если условие истинно, то будут выполнены операторы в блоке **if**, если ложно, то управление будет передано следующему блоку **else-if** для проверки следующего условия. Если все условия ложны, то будут выполнены операторы в блоке **else**

```
if (условие 1) {
    // Сделай это;
} else if (условие 2) {
    // Или это;
    // И это;
} else if (условие N) {
    // Или хотя бы это;
} else {
    // Тогда сделай так;
}
```



Операторы ветвления

Оператор if-else-if-else позволяет проверить несколько взаимоисключающих условий

```
int a = 0;
if (a > 0) {
    std::cout << "a is positive" << std::endl;
} else if (a < 0) {
    std::cout << "a is negative" << std::endl;
} else {
    std::cout << "a is zero" << std::endl;
}
```

Логические операторы

Логические операторы определены для типа **Boolean** и позволяют вычислять простые или составные логические выражения

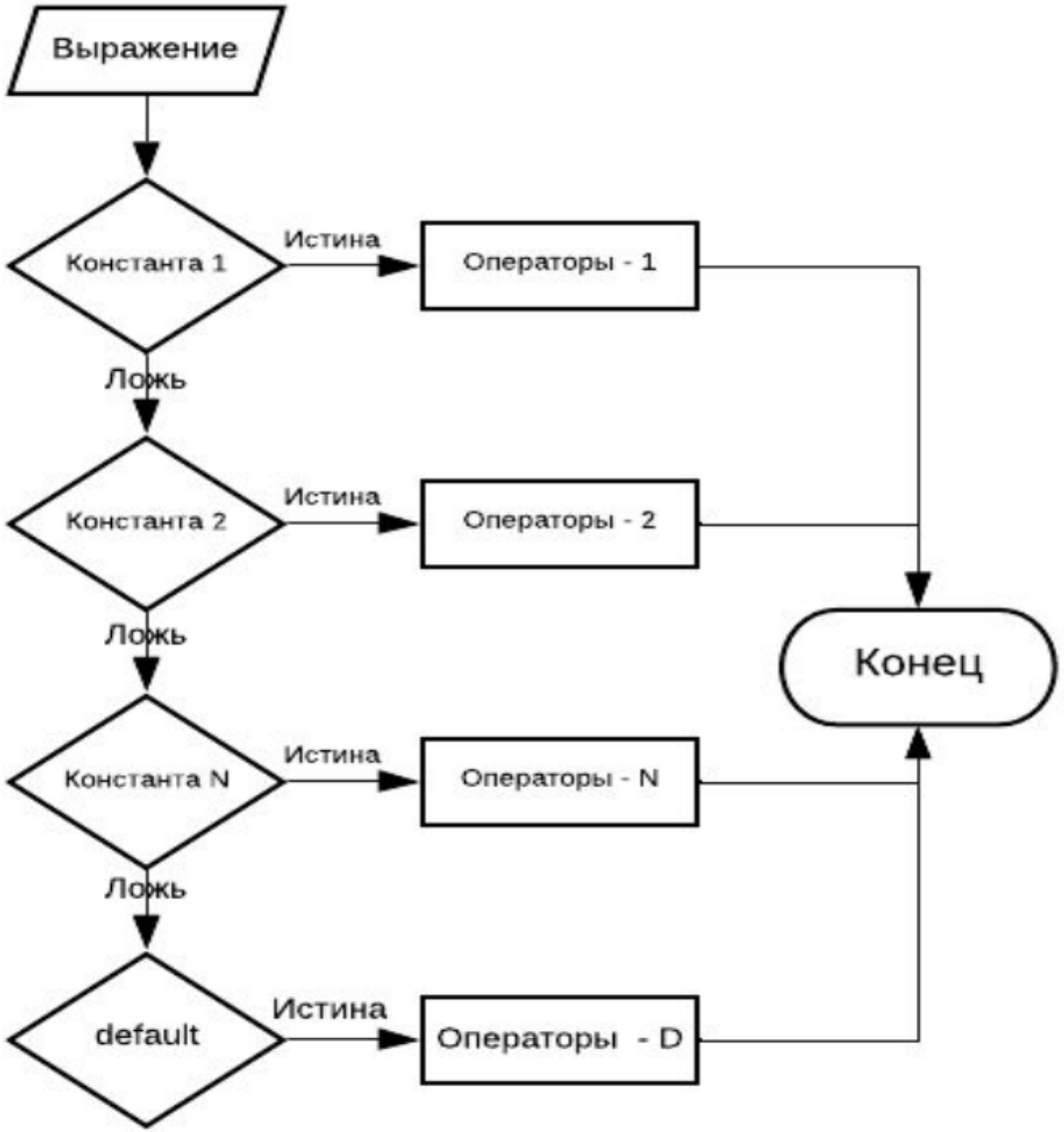
Оператор	Назначение
!	Оператор **отрицание** возвращает **true**, если операнд **false**
&&	Оператор **И** возвращает **true**, если оба операнда **true**
||	Оператор **ИЛИ** возвращает **true**, если хотя бы один из двух операндов **true**

```
if (b && c) {
    std::cout << "a / (b * c) = " << a / (b * c) << std::endl;
} else
    std::cout << "Error! Denominator is equal to zero" << std::endl;
}
```

Операторы ветвления

Оператор switch

Оператор **switch** последовательно проверяет результат выражения на соответствие одному из значений. При первом совпадении с константой будут выполнены операторы, соответствующие совпадающему значению, после чего управление будет передано дальше. Если ни одна из констант не совпала со значением выражения, то будут выполнены операторы из блока **default**.



Операторы ветвления

Оператор switch

Оператор **switch** выбирает один из блоков для выполнения. Оператор последовательно сравнивает операнд с вариантами значений и при совпадении значения передает управление оператору блока следующего за меткой **case** с совпавшим значением. Если нет ни одного подходящего блока **case**, передает управление оператору блока **default**.

```
int num = 3;
switch(num) {
    case 1:
        std::cout << "One" << std::endl;
        break;
    case 2:
        std::cout << "Two" << std::endl;
        break;
    case 3:
        std::cout << "Three" << std::endl;
        break;
    default:
        std::cout << "Other number" << std::endl;
        break;
}
```

Операторы ветвления

Правила оператора switch

В **case** обязательно использовать интегральный тип данных (т.е. char, short, int, long, long long или enum). Неинтегральный тип или тип с плавающей точкой использоваться не могут.

Константы не могут повторяться, то есть в **case** всегда определены исключающие друг друга значения.

Каждый раздел операторов **switch** может содержать одну или несколько меток **case**.

```
int num = 3;
switch(num) {
    case 1:
    case 3:
        std::cout << “Odd” << std::endl;
        break;
    case 2:
    case 4:
        std::cout << “Even” << std::endl;
        break;
    default:
        std::cout << “Parity error” <<
                                    std::endl;
        break;
}
```

Операторы ветвления

Тернарный оператор

Оператор **?:** выполняет Выражение 1 если условие Истинно, иначе выполняет Выражение 2.

В общем виде записывается следующим образом:

```
Условие ? Выражение 1 : Выражение 2
```

Запомнить правила вычисления тернарного оператора можно как:

```
Это условие истинно ? да : нет
```

Операторы ветвления

Тернарный оператор ?: позволяет лаконично записывать условный код:

```
std::string result = a >= 0 ? "positive" : "negative";
std::cout << result << std::endl;
```

Тот же код, написанный с использованием оператора if-else:

```
int a = 0;
std::string result;
if(a >= 0) {
    result = "positive";
} else {
    result = "negative";
}
std::cout << result << std::endl;
```

Операторы ветвления

Что будет выведено на экран?

```
int a = 6;
if(a > -10 && a < 10) {
    switch(a % 2) {
        case 0:
            std::cout << (a < 0 ? "negative " : "") << "even" << std::endl;
            break;
        default:
            std::cout << (a < 0 ? "negative " : "") << "odd" << std::endl;
            break;
    }
}
```

КАИ
Крылья для жизни!

Спасибо за внимание!