УНИВЕРСИТЕТ СИНЕРГИЯ

РАЗРАБОТКА НА C++

Урок 3.
Булевый тип данных и условия

Булевый тип данных

Булевый тип данных (bool) в C++ предназначен для хранения логических значений. Он может принимать только два значения: true (истина) и false (ложь).

Пример объявления переменной булевого типа:

```
bool isTrue = true;
bool isFalse = false;
```

Любое значение, которое будет положено в переменную bool **будет превращаться в true**.
Исключения это следующие **значения для false**: 0, NULL, nullptr, false, “” (это пустая строка)

```
bool a = 1;       //true
bool b = 15;      //true
bool c = -12562; //true
bool d = ‘s’;     //true
bool e = 0;       //false
bool f = NULL;    //false
```

Логические операторы

Логические операторы позволяют комбинировать булевы значения и выполнять логические операции:

- оператор AND (&&) возвращает true только если оба операнда равны true;
- оператор OR (||) возвращает true, если хотя бы один из операндов равен true;
- оператор NOT (!) инвертирует значение булевого операнда.

Примеры:

```
bool a = true;
bool b = false;
bool result1 = a && b; // result1 будет равен false
bool result2 = a || b; // result2 будет равен true
bool result3 = !a;     // result3 будет равен false
```

Приоритет операций

Приоритет операций с булевыми данными:

1. NOT (!)
2. AND (&&)
3. OR (||).

При работе с логическими операторами можно совершить ошибку в определении приоритета операций, что может привести к неправильным результатам. Используйте скобки, чтобы явно указать порядок выполнения операций при необходимости.

Приоритет операций

Давайте рассмотрим пример:

```
bool a = true;
bool b = false;
bool c = true;

bool result = a || b && c;
```

Здесь есть два логических оператора: OR (||) и AND (&&). Многие новички могут допустить ошибку, думая, что оператор OR (||) имеет более высокий приоритет, и поэтому выражение a || b && c будет интерпретироваться как (a || b) && c.

Однако в C++ оператор AND (&&) имеет более высокий приоритет, чем OR (||). Поэтому выражение будет интерпретировано как a || (b && c).

Приоритет операций

Чтобы избежать подобных ошибок, всегда рекомендуется использовать скобки, явно указывая порядок выполнения операций:

```
bool result = (a || b) && c;
```

В этом случае будет выполнено сначала логическое OR, а затем логическое AND, что может соответствовать вашим ожиданиям и предотвратить ошибки в приоритете операций.

Решим задачи

Дано:

```
bool a = true;
bool b = false;
```

Что вернут данные операции?

```
!a                    = ?
!b && a               = ?
!b || !a              = ?
(!b || a) && a        = ?
(b || !a) && a || !b  = ?
```

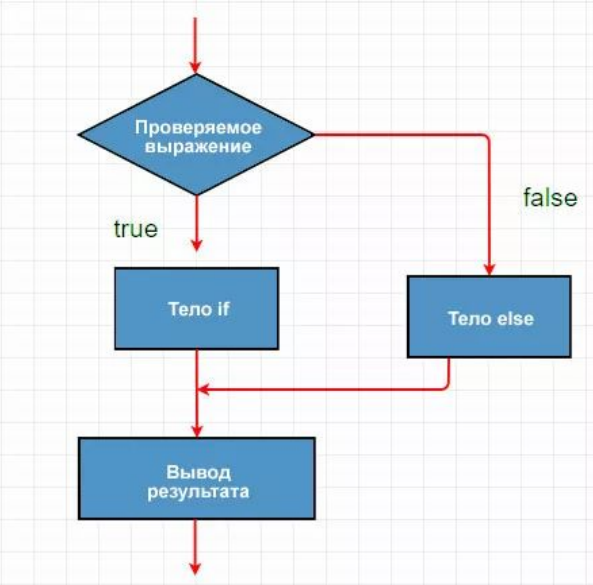
Структура условной конструкции

Условная конструкция в C++ состоит из двух частей: ветки «if» и ветки «else». После слова «if» описывается проверяемое условие, далее следуют команды, которые выполняются, если условие истинно.
Ветка «else» идет следом за веткой «if». В ветке «else» размещают команды, которые выполняются если проверяемое условие всё-таки ложно.

Синтаксис условной конструкции:

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

Схема работы условных конструкций



Условия в конструкции «if»

Давайте представим, что нам нужно проверить, является ли значение переменной `a` четным. На первый взгляд, это можно сделать так:

```
a = 4;
if (a/2) {
    cout<<”Число четное”
} else {
    cout<<”Число нечетное”
}
```

Но выражение `a/2` выдаст в ответ 2 (4 / 2 = 2). Ответ в виде числа 2 не дает понять, `a` - это четное число или нет.

Проверяемое выражение (условие) должно возвращать ответ в виде булевого значения. Чтобы проверить, является ли значение переменной `a` четным, подойдет выражение `a%2==0`. Выражение `a%2==0` содержит знак `==`, оно вернет `True` или `False`. Если условная конструкция получит при проверке условия `True`, программа продолжит работу по ветке «if». Если же условная конструкция получит при проверке условия `False`, программа пойдет по ветке «else».

Логические операторы в «if»

Выше мы рассматривали логические операторы NOT (!), AND (&&), OR (||). Они могут использоваться в условных конструкциях. Например, в случае, когда нам нужно проверить, является ли число двузначным:

```
 a = 24;
if (a>9 && a<100) {
    cout<<”Число двузначное”
} else {
    cout<<”Число не является двузначным”
}
```

Конструкция «if» без «else»

Бывают ситуации, когда программа при ложном условии ничего делать не должна. За действия, когда условие не выполняется, отвечает ветка «else». В таком случае эту ветку можно просто не писать:

```
 a = 24;
if (a>9 && a<100) {
    cout<<”Число двузначное”
}
```

Этот фрагмент программы будет работать исправно. Если число в переменной `a` будет двузначным, программа по-прежнему будет идти по ветке «if». Если же в переменную `a` попадет трехзначное число, программа должна пойти по ветке «else». Но так как этой ветки нет, программа не пойдет ни по одной из веток — ошибки не случится.

Вложенные условия

Задача: проверить, является ли число положительным, четным и меньшим 10. Вложенные условия позволят нам легко выполнить эту задачу:

```
int number = 8;

if (number > 0) {
    // Внешнее условие: число положительное
    if (number % 2 == 0) {
        // Внутреннее условие: число четное
        if (number < 10) {
            // Второе внутреннее условие: число меньше 10
            cout << "Число удовлетворяет всем условиям." <<
endl;
        } else {
            cout << "Число не меньше 10." << endl;
        }
    } else {
        cout << "Число не четное." << endl;
    }
} else {
    cout << "Число не положительное." << endl;
}
```

Как это работает:

1. Внешнее условие проверяет, является ли число положительным. Если это условие ложно, программа переходит к блоку кода после else внешнего условия и выводит сообщение "Число не положительное."
2. Если внешнее условие истинно, программа переходит к внутреннему условию. Внутреннее условие проверяет, является ли число четным. Если это условие ложно, программа переходит к блоку кода после else внутреннего условия и выводит сообщение "Число не четное."
3. Если и внешнее и внутреннее условия истинны, программа переходит ко второму внутреннему условию. Оно проверяет, меньше ли число 10. Если это условие ложно, программа выводит сообщение "Число не меньше 10." Если условие истинно, программа выводит сообщение "Число удовлетворяет всем условиям."

Множественные условия

Задача: оценить погоду на основе температуры.
Чтобы ее реализовать можно использовать уже известные нам вложенные условия:

```
int temperature = 25;

if (temperature < 0) {
    cout << "Очень холодно!" << endl;
} else {
    if (temperature >= 0 && temperature < 10) {
        cout << "Холодно." << endl;
    } else {
        if (temperature >= 10 && temperature < 20) {
            cout << "Прохладно." << endl;
        } else {
            if (temperature >= 20 && temperature < 30) {
                cout << "Тепло." << endl;
            } else {
                cout << "Жарко!" << endl;
            }
        }
    }
}
```

Выглядит громоздко.

Множественные условия

Этот код можно оптимизировать с помощью множественных условий. Вот так:

```
int temperature = 25;

if (temperature < 0) {
    cout << "Очень холодно!" << endl;
} else if (temperature >= 0 && temperature < 10) {
    cout << "Холодно." << endl;
} else if (temperature >= 10 && temperature < 20) {
    cout << "Прохладно." << endl;
} else if (temperature >= 20 && temperature < 30) {
    cout << "Тепло." << endl;
} else {
    cout << "Жарко!" << endl;
}
```

Множественные условия представляют собой «склеенные» условные конструкции с одной общей веткой «else». Здесь каждое условие проверяется, только если не выполнилось предыдущее. Если ни одно условие не сработает, выполнится общая ветка «else».

Вложенные или множественные условия?

Когда условия взаимосвязаны и зависят друг от друга, удобнее использовать вложенные условия. Они позволят легко выразить сложные логические зависимости между условиями.

Шаблон конструкции:

```
if (условие1) {
    if (условие2) {
        // код
    } else {
        // код
    }
} else {
    // код
}
```

Вложенные или множественные условия?

Если условия, которые вы хотите проверить, независимы друг от друга и выполняются независимо, множественные условия могут быть более подходящим способом их проверки.

Шаблон конструкции:

```
if (условие1) {
    // код
} else if (условие2) {
    // код
} else if (условие3) {
    // код
} else {
    // код
}
```