

Ravesli [Ravesli](#)


- [Уроки по C++](#)
- [OpenGL](#)
- [SFML](#)
- [Qt5](#)
- [RegExр](#)
- [Ассемблер](#)
- [Купить .PDF](#)


Урок №64. Операторы условного ветвления if/else

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 15 Сен 2020 |

 38071

[14](#)

На этом уроке мы рассмотрим операторы условного ветвления if/else, а также то, как их можно использовать.

Оглавление:

1. [Условные ветвления if/else](#)
2. [Использование нескольких операций в ветвлениях if/else](#)
3. [Неявное указание блоков](#)
4. [Связывание стейтментов if](#)
5. [Вложенные ветвления if/else](#)
6. [Использование логических операторов в ветвлениях if/else](#)
7. [Основные использования ветвлений if/else](#)
8. [Нулевые стейтменты](#)

Условные ветвления if/else

Самыми простыми условными ветвлениями в языке C++ являются стейтменты if/else. Они выглядят следующим образом:

```
if (выражение)
    стейтмент1
```

Либо так:

```
if (выражение)
    стейтмент1
else
    стейтмент2
```

выражение называется **условием** (или «**условным выражением**»). Если результатом выражения является true (любое ненулевое значение), то выполняться будет стейтмент1. Если же результатом выражения является false (0), то выполняться будет стейтмент2. Например:

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Enter a number: ";
6      int a;
7      std::cin >> a;
8
9      if (a > 15)
10         std::cout << a << " is greater than 15\n";
11     else
12         std::cout << a << " is not greater than 15\n";
13
14     return 0;
15 }
```

Использование нескольких операций в ветвлениях if/else

Оператор if выполняет *только одну* операцию, если выражение является true, и также *только одну* операцию else, если выражение — false. Чтобы выполнить несколько операций подряд, используйте **блок стейтментов**:

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Enter a number: ";
6      int a;
7      std::cin >> a;
8
9      if (a > 15)
10     {
11         // Обе операции будут выполнены, если a > 15
12         std::cout << "You entered " << a << "\n";
13         std::cout << a << " is greater than 15\n";
14     }
15     else
16     {
```

```
17 // Обе операции будут выполнены, если a <= 15
18 std::cout << "You entered " << a << "\n";
19 std::cout << a << " is not greater than 15\n";
20 }
21
22 return 0;
23 }
```

Неявное указание блоков

Если программист не указал скобки для блока стейтментов if или else, то компилятор неявно сделает это за него. Таким образом, следующее:

```
if (выражение)
    стейтмент1
else
    стейтмент2
```

Будет выполняться как:

```
if (выражение)
{
    стейтмент1
}
else
{
    стейтмент2
}
```

По сути, это не имеет значения. Однако начинающие программисты иногда пытаются сделать что-то вроде следующего:

```
1 #include <iostream>
2
3 int main()
4 {
5     if (1)
6         int a = 4;
7     else
8         int a = 5;
9
10    std::cout << a;
11
12    return 0;
13 }
```

Программа не скомпилируется, и в итоге мы получим ошибку, что идентификатор a не определен. А произойдет это из-за того, что программа будет выполняться следующим образом:

```
1 #include <iostream>
2
3 int main()
4 {
5     if (1)
6     {
7         int a = 4;
8     } // переменная a уничтожается здесь
9     else
10    {
11        int a = 5;
12    } // переменная a уничтожается здесь
13
14    std::cout << a; // переменная a здесь не определена
15
16    return 0;
17 }
```

В этом контексте становится понятным, что переменная *a* имеет локальную область видимости и уничтожается в конце блока, в котором выполняется её инициализация. И, когда мы дойдем до строчки с std::cout, переменная *a* уже перестанет существовать.

Связывание стейтментов if

Стейтменты if/else можно использовать в связке:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Enter a number: ";
6     int a;
7     std::cin >> a;
8
9     if (a > 15)
10        std::cout << a << " is greater than 15\n";
11    else if (a < 15)
12        std::cout << a << " is less than 15\n";
13    else
14        std::cout << a << " is exactly 15\n";
15
16    return 0;
17 }
```

Вложенные ветвления if/else

Одни стейтменты `if` могут быть вложены в другие стейтменты `if`:

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Enter a number: ";
6      int a;
7      std::cin >> a;
8
9      if (a > 15) // внешний оператор if
10         // Это плохой способ написания вложенных стейтментов if
11         if (a < 25) // внутренний оператор if
12             std::cout << a << " is between 15 and 25\n";
13
14         // К какому if относится следующий else?
15         else
16             std::cout << a << " is greater than or equal to 25\n";
17
18     return 0;
19 }
```

Обратите внимание, в программе, приведенной выше, мы можем наблюдать **потенциальную ошибку двусмысленности оператора `else`**. К какому `if` относится оператор `else`: к внешнему или к внутреннему?

Дело в том, что оператор `else` всегда относится к последнему незакрытому оператору `if` в блоке, в котором находится сам `else`. Т.е. в программе, приведенной выше, `else` относится к внутреннему `if`.

Чтобы избежать таких вот неоднозначностей при вложенности операторов условного ветвления, рекомендуется использовать блоки стейтментов (указывать скобки). Например, вот та же программа, приведенная выше, но уже без двусмысленности:

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Enter a number: ";
6      int a;
7      std::cin >> a;
8
9      if (a > 15)
10     {
11         if (a < 25)
12             std::cout << a << " is between 15 and 25\n";
13         else // относится к внутреннему оператору if
14             std::cout << a << " is greater than or equal to 25\n";
15     }
16
17     return 0;
```

```
18 }
```

Теперь понятно, что оператор `else` относится к внутреннему оператору `if`. Использование скобок также позволяет явно указать привязку `else` к внешнему стейтменту `if`:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Enter a number: ";
6     int a;
7     std::cin >> a;
8
9     if (a > 15)
10    {
11        if (a < 25)
12            std::cout << a << " is between 15 and 25\n";
13    }
14    else // относится к внешнему оператору if
15        std::cout << a << " is less than 15\n";
16
17    return 0;
18 }
```

Используя блоки стейтментов, мы уточняем, к какому `if` следует прикреплять определенный `else`. Без блоков оператор `else` будет прикрепляться к ближайшему незакрытому оператору `if`.

Использование логических операторов в ветвлениях if/else

Также вы можете проверить сразу несколько условий в ветвлениях `if/else`, используя [логические операторы](#):

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Enter an integer: ";
6     int a;
7     std::cin >> a;
8
9     std::cout << "Enter another integer: ";
10    int b;
11    std::cin >> b;
12
13    if (a > 0 && b > 0) // && - это логическое И. Проверяем, являются ли оба условия истинными
14        std::cout << "Both numbers are positive\n";
15    else if (a > 0 || b > 0) // || - это логическое ИЛИ. Проверяем, является ли истинным хотя бы одно из условий
16        std::cout << "At least one number is positive\n";
17    else
18        std::cout << "Both numbers are non-positive\n";
19
20    return 0;
21 }
```

```
16         std::cout << "One of the numbers is positive\n";
17     else
18         std::cout << "Neither number is positive\n";
19
20     return 0;
21 }
```

Основные использования ветвлений if/else

Ветвления if/else активно используются для проверки ошибок. Например, чтобы вычислить квадратный корень значения, параметр, который передается в функцию для вычисления, — обязательно должен быть положительным:

```
1  #include <iostream>
2  #include <cmath> // для функции sqrt()
3
4  void printSqrt(double value)
5  {
6      if (value >= 0.0)
7          std::cout << "The square root of " << value << " is " << sqrt(value) << "\n";
8      else
9          std::cout << "Error: " << value << " is negative\n";
10 }
```

Также операторы if используют для **ранних возвратов** — когда функция возвращает управление обратно в caller еще до завершения выполнения самой функции. В программе, приведенной ниже, если значением параметра является отрицательное число, то функция сразу же возвращает в caller символьную константу или перечислитель в качестве кода ошибки:

```
1  #include <iostream>
2
3  enum class ErrorCode
4  {
5      ERROR_SUCCESS = 0,
6      ERROR_NEGATIVE_NUMBER = -1
7  };
8
9  ErrorCode doSomething(int value)
10 {
11     // Если параметром value является отрицательное число,
12     if (value < 0)
13         // то сразу же возвращаем код ошибки
14         return ErrorCode::ERROR_NEGATIVE_NUMBER;
15
16     // Что-нибудь делаем
17
18     return ErrorCode::ERROR_SUCCESS;
19 }
```

```
20
21 int main()
22 {
23     std::cout << "Enter a positive number: ";
24     int a;
25     std::cin >> a;
26
27     if (doSomething(a) == ErrorCode::ERROR_NEGATIVE_NUMBER)
28     {
29         std::cout << "You entered a negative number!\n";
30     }
31     else
32     {
33         std::cout << "It worked!\n";
34     }
35
36     return 0;
37 }
```

Ветвления `if/else` также обычно используют для выполнения простых математических операций. Например, рассмотрим функцию `min()`, которая возвращает минимальное из 2-х чисел:

```
1 int min(int a, int b)
2 {
3     if (a > b)
4         return b;
5     else
6         return a;
7 }
```

Эта функция настолько проста, что её можно записать с помощью [условного тернарного оператора](#):

```
1 int min(int a, int b)
2 {
3     return (a > b) ? b : a;
4 }
```

Нулевые стейтменты

Также в C++ можно не указывать основную часть оператора `if`. Такие стейтменты называются **нулевыми стейтментами** (или «*null-стейтментами*»). Объявить их можно, используя точку с запятой вместо выполняемой операции. В целях улучшения читабельности кода, точка с запятой нулевого стейтмента обычно пишется с новой строки. Таким образом, мы *явно указываем*, что хотим использовать `null-стейтмент`, уменьшая вероятность не заметить его использования:

```
1 if (a > 15)
2     ; // это нулевой стейтмент
```


Хотя нулевые стейтменты редко используются в сочетании с оператором `if`, но, из-за неосторожности, это может привести к проблемам. Рассмотрим следующий фрагмент кода:

```
1 if (a == 0);  
2   a = 1;
```

В вышеприведенном примере мы случайно указали точку с запятой в конце оператора `if`. Эта неосмотрительность приведет к тому, что код будет выполняться следующим образом:

```
1 if (a == 0)  
2   ; // точка с запятой указывает, что это нулевой стейтмент  
3 a = 1; // и эта строка выполнится в любом случае!
```

Предупреждение: Всегда проверяйте, не «закрыли» ли вы случайно оператор `if` точкой с запятой.

Оценить статью:

★★★★★ (265 оценок, среднее: 4,95 из 5)



[← Урок №63. Операторы управления потоком выполнения программ](#)

[Урок №65. Оператор switch](#)



Комментариев: 14



1. [Ахмадиёр:](#)
[28 марта 2020 в 19:17](#)

```
1 for(int i=0;i<1000;i++)  
2 cout<<"thank you very much Yuriy"<<endl;
```

[Ответить](#)



1. [Валера:](#)
[30 марта 2020 в 19:33](#)

А компилировать кто будет ? петухон 7

[Ответить](#)



2. [Андрей:](#)
[11 февраля 2020 в 05:55](#)

А как же присвоить значение переменной по определенному условию и вывести из ветвления для дальнейшей обработки?

[Ответить](#)



1. *Константин:*

[10 мая 2020 в 11:09](#)

Можно объявить переменную перед if, например

```
...  
int a;  
If(условие)  
a = 3;  
  
else  
a = 5
```

[Ответить](#)



3. *Алексей:*

[17 июля 2019 в 12:59](#)

Прямо второе дыхание открылось.

Честно говоря — не люблю перечислители. Пока что не видел большого применения.

[Ответить](#)



4. *Константин:*

[6 апреля 2019 в 04:27](#)

Юра, в стейтменте

```
1 | for (i = 0; example[i] != '\0'; i++)
```

'\0' — что означает?

[Ответить](#)



1. *Анастасия:*

[11 июня 2019 в 20:30](#)

Я не Юра, но попробую ответить.

\0 — это завершающий нулевой символ, показывающий, что строка на этом заканчивается. Это обязательный последний символ для символьных массивов, дальше него массив не выводится, а без него вывод массива приведёт к ошибке.

Например, если есть массив {'C', 'A', 'Ш', 'A', '/0', '!'}, и мы выводим его на экран, то получим только САША

[Ответить](#)



1. Константин:

[26 января 2020 в 11:01](#)

а прямой ли, обратный ли слэш — без разницы?!

[Ответить](#)



5. master114:

[7 мая 2018 в 16:04](#)

Спасибо за проделанную работу.

Ценность не только в переводе хорошего учебника (хотя это титанический труд) но и в поддержке студентов. Отвечаете на вопросы, комментируете тонкости, просматриваете ответы на задачи и так далее. Это очень важно!!!

[Ответить](#)

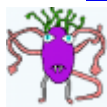


1. Юрий:

[7 мая 2018 в 18:35](#)

Сам студент и понимаю, что так и должно быть. Если есть возможность — даю фидбек 😊

[Ответить](#)



6. Человек:

[15 марта 2018 в 17:04](#)

Не уточните, зачем вообще используются стейтменты null? Программа же просто проигнорирует эти две строки, разве нет?

[Ответить](#)



1. Юрий:

[15 марта 2018 в 21:50](#)

Стейтменты null ничего не делают, но существуют из-за синтаксических причин. Наиболее часто они используются в качестве заполнителей в итерационных операциях, циклах, ветвлениях if. Например, вы хотите написать свою функцию определения длины строки:

```
1 #include "stdafx.h"
2 #include <iostream>
3 #include <string>
```

```
4
5 int myStrLen(std::string example)
6 {
7     int i;
8     for (i = 0; example[i] != '\0'; i++)
9         /* null statement */;
10    return i;
11 }
12
13 int main()
14 {
15     std::string name("Sasha");
16     std::cout << myStrLen(name);
17 }
18 }
```

Стейтмент `null` здесь служит чисто для заполнения тела цикла `for` (тело должно быть обязательно). `return i`; поместить в тело цикла `for` нельзя, так как тогда мы получим просто 0.

Еще вариант применения:

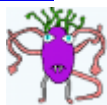
```
1 if (condition1)
2     if (condition2)
3         doSomething();
4     else
5         ; /* null statement */
6 else
7     doSomethingElse();
```

В этом случае внутренний оператор `else` и стейтмент `null` удерживают внешний `else` от привязки к внутреннему `if`.

Также еще может быть случай, когда вы хотите сохранить прежний код, закомментировав его, тем самым сохраняя возможность легкого восстановления, если он потребуется:

```
1 if (x <= 1)
2     /*std::cout << "No arguments";*/ ;
3 else
4     std::cout << "Argument: " << x;
```

[Ответить](#)



1. Человек:

[16 марта 2018 в 15:00](#)

Все, понял, спасибо большое

[Ответить](#)



1. *Юрий:*
[16 марта 2018 в 15:11](#)

Пожалуйста 😊

Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены *

Имя *

Email *






Комментарий

☐ Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию

☐ Получать уведомления о новых комментариях по электронной почте. Вы можете [подписаться](#) без комментирования.

[TELEGRAM](#)  [КАНАЛ](#)
[ПАБЛИК](#) 

ТОП СТАТЬИ

-  [Словарь программиста. Сленг, который должен знать каждый кодер](#)
-  [Урок №1. Введение в программирование](#)
-  [70+ бесплатных ресурсов для изучения программирования](#)
-  [Урок №1: Введение в создание игры «SameGame» на C++/MFC](#)
-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

- [Ravesli](#)
- - [О проекте/Контакты](#) -

- - [Пользовательское Соглашение](#) -
- - [Все статьи](#) -
- Copyright © 2015 - 2020