5.2 — If statements

BY ALEX ON JUNE 21ST, 2007 | LAST MODIFIED BY ALEX ON JANUARY 23RD, 2020

The most basic kind of conditional branch in C++ is the **if statement**. An *if statement* takes the form:

```
if (expression)
    statement


or

if (expression)
    statement
else
    statement2
```

The expression is called a **conditional expression**. If the expression evaluates to true (non-zero), the statement executes. If the expression evaluates to false, the *else statement* is executed if it exists.

Here is a simple program that uses an *if statement*:

```
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;

    if (x > 10)
        std::cout << x << "is greater than 10\n";
    else
        std::cout << x << "is not greater than 10\n";

    return 0;
}
```

Using if with multiple statements

Note that the *if statement* only executes a single statement if the expression is true, and the *else* only executes a single statement if the expression is false. In order to execute multiple statements, we can use a block:

```
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;

    if (x > 10)
    {
        // both statements will be executed if x > 10
        std::cout << "You entered " << x << "\n";
        std::cout << x << "is greater than 10\n";
    }
    else
```

```
    {
        // both statements will be executed if x <= 10
        std::cout << "You entered " << x << "\n";
        std::cout << x << "is not greater than 10\n";
    }

    return 0;
}
```

Implicit blocks

If the programmer does not declare a block in the statement portion of an *if statement* or *else statement*, the compiler will implicitly declare one. Thus:

```
if (expression)
    statement
else
    statement2
```

is actually the equivalent of:

```
if (expression)
{
    statement
}
else
{
    statement2
}
```

Most of the time, this doesn't matter. However, new programmers sometimes try to do something like this:

```
#include <iostream>

int main()
{
    if (true)
        int x = 5;
    else
        int x = 6;

    std::cout << x;

    return 0;
}
```

This won't compile, with the compiler generating an error that identifier x isn't defined. This is because the above example is the equivalent of:

```
#include <iostream>

int main()
{
    if (true)
    {
        int x = 5;
    } // x destroyed here
    else
    {
```

```
        int x = 6;
    } // x destroyed here

    std::cout << x; // x isn't defined here

    return 0;
}
```

In this context, it's clearer that variable x has block scope and is destroyed at the end of the block. By the time we get to the std::cout line, x doesn't exist.

Chaining if statements

It is possible to chain if-else statements together:

```
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;

    if (x > 10)
        std::cout << x << "is greater than 10\n";
    else if (x < 10)
        std::cout << x << "is less than 10\n";
    else
        std::cout << x << "is exactly 10\n";

    return 0;
}
```

The above code executes identically to the following (which may be easier to understand):

```
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;

    if (x > 10)
        std::cout << x << "is greater than 10\n";
    else
    {
        if (x < 10)
            std::cout << x << "is less than 10\n";
        else
            std::cout << x << "is exactly 10\n";
    }

    return 0;
}
```

First, x > 10 is evaluated. If true, then the "is greater" output executes and we're done. Otherwise, the else statement executes. That else statement is a nested if statement. So we then check if x < 10. If true, then the "is less than" output executes and we're done. Otherwise, the nested else statement executes. In that case, we print the "is exactly" output, and we're done.

In practice, we typically don't nest the chained-ifs inside blocks, because it makes the statements harder to read (and the code quickly becomes nested if there are lots of chained if statements).

Here's another example:

```
#include <iostream>

int main()
{
    std::cout << "Enter a positive number between 0 and 9999: ";
    int x;
    std::cin >> x;

    if (x < 0)
        std::cout << x << " is negative\n";
    else if (x < 10)
        std::cout << x << " has 1 digit\n";
    else if (x < 100)
        std::cout << x << " has 2 digits\n";
    else if (x < 1000)
        std::cout << x << " has 3 digits\n";
    else if (x < 10000)
        std::cout << x << " has 4 digits\n";
    else
        std::cout << x << " was larger than 9999\n";

    return 0;
}
```

Nesting if statements

It is also possible to nest if statements within other if statements:

```
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;

    if (x >= 10) // outer if statement
        // it is bad coding style to nest if statements this way
        if (x <= 20) // inner if statement
            std::cout << x << "is between 10 and 20\n";

        // which if statement does this else belong to?
        else
            std::cout << x << "is greater than 20\n";

    return 0;
}
```

The above program introduces a source of potential ambiguity called a **dangling else** problem. Is the *else statement* in the above program matched up with the outer or inner *if statement*?

The answer is that an *else statement* is paired up with the last unmatched *if statement* in the same block. Thus, in the program above, the *else* is matched up with the inner *if statement*.

To avoid such ambiguities when nesting complex statements, it is generally a good idea to enclose the statement within a block. Here is the above program written without ambiguity:

```
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;

    if (x >= 10)
    {
        if (x <= 20)
            std::cout << x << "is between 10 and 20\n";
        else // attached to inner if statement
            std::cout << x << "is greater than 20\n";
    }

    return 0;
}
```

Now it is much clearer that the *else statement* belongs to the inner *if statement*.

Encasing the inner *if statement* in a block also allows us to explicitly attach an *else* to the outer *if statement*:

```
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;

    if (x >= 10)
    {
        if (x <= 20)
            std::cout << x << "is between 10 and 20\n";
    }
    else // attached to outer if statement
        std::cout << x << "is less than 10\n";

    return 0;
}
```

The use of a block tells the compiler that the *else statement* should attach to the *if statement* before the block. Without the block, the *else statement* would attach to the nearest unmatched *if statement*, which would be the inner *if statement*.

Using logical operators with if statements

You can also have if statements check multiple conditions together by using the logical operators (covered in section **3.6 -- logical operators**):

```
#include <iostream>

int main()
{
    std::cout << "Enter an integer: ";
    int x;
    std::cin >> x;

    std::cout << "Enter another integer: ";
    int y;
    std::cin >> y;
```

```
12
13      if (x > 0 && y > 0) // && is logical and -- checks if both conditions are true
14          std::cout << "both numbers are positive\n";
15      else if (x > 0 || y > 0) // || is logical or -- checks if either condition is true
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  }
```

Common uses for if statements

If statements are commonly used to do error checking. For example, to calculate a square root, the value passed to the square root function should be a non-negative number:

```
1   #include <iostream>
2   #include <cmath> // for 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  }
```

If statements can also be used to do **early returns**, where a function returns control to the caller before the end of the function. In the following program, if the parameter value is negative, the function returns a symbolic constant or enumerated value error code to the caller right away.

```
1   #include <iostream>
2
3   enum class ErrorCode
4   {
5       SUCCESS = 0,
6       NEGATIVE_NUMBER = -1
7   };
8
9   ErrorCode doSomething(int value)
10  {
11      // if value is a negative number
12      if (value < 0)
13         // early return an error code
14          return ErrorCode::NEGATIVE_NUMBER;
15
16      // Do whatever here
17
18      return ErrorCode::SUCCESS;
19  }
20
21  int main()
22  {
23      std::cout << "Enter a positive number: ";
24      int x;
25      std::cin >> x;
26
27      if (doSomething(x) == ErrorCode::NEGATIVE_NUMBER)
28      {
29          std::cout << "You entered a negative number!\n";
30      }
31      else
```

```
    {
        std::cout << "It worked!\n";
    }

    return 0;
}
```

If statements are also commonly used to do simple math functionality, such as a min() or max() function that returns the minimum or maximum of its parameters:

```
int min(int x, int y)
{
    if (x > y)
        return y;
    else
        return x;
}
```

Note that this last function is so simple, it can also be written using the conditional operator (?:):

```
int min(int x, int y)
{
    return (x > y) ? y : x;
}
```

Null statements

It is possible to omit the statement part of an *if statement*. A statement with no body is called a **null statement**, and it is declared by using a single semicolon in place of the statement. For readability purposes, the semicolon of a *null statement* is typically placed on its own line. This indicates that the use of a *null statement* was intentional, and makes it harder to overlook the use of the *null statement*.

```
if (x > 10)
    ; // this is a null statement
```

Null statements are typically used when the language requires a statement to exist but the programmer doesn't need one. We'll see examples of intentional null statements later in this chapter, when we cover loops.

Null statements are rarely used in conjunction with *if statements*. However, they often unintentionally cause problems for new or careless programmers. Consider the following snippet:

```
if (x == 0);
    x = 1;
```

In the above snippet, the user accidentally put a semicolon on the end of the *if statement*. This unassuming error actually causes the above snippet to execute like this:

```
if (x == 0)
    ; // the semicolon acts as a null statement
x = 1; // and this line always gets executed!
```

Warning: Make sure you don't accidentally "terminate" your if statements with a semicolon.

Operator== vs Operator= inside the conditional

Just a reminder that inside your if statement conditional, you should be using operator== when testing for equality, not operator= (which is assignment). Consider the following program:

```
#include <iostream>

int main()
{
    std::cout << "Enter 0 or 1: ";
```

```
    int x;
    std::cin >> x;
    if (x = 0) // oops, we used an assignment here instead of a test for equality
        std::cout << "You entered 0";
    else
        std::cout << "You entered 1";

    return 0;
}
```

This program will compile and run, but will always produce the result "You entered 1" even if you enter 0. This happens because x = 0 first assigns 0 to x, then evaluates to the value 0, which is boolean false. Since the conditional is always false, the else statement always executes.

Init-statements

If you need a variable in an `if`-statement, but not outside, you can use an *init-statement* before the condition.

```
#include <iostream>
#include <string>

int main()
{
    std::string firstName{};
    std::string lastName{};

    std::cout << "First name: ";
    std::cin >> firstName;

    std::cout << "Last name: ";
    std::cin >> lastName;

    if (std::string fullName{ firstName + ' ' + lastName }; fullName.length() > 20)
    {
        std::cout << '"' << fullName << "\"is too long!\n";
    }
    else
    {
        std::cout << "Your name is " << fullName << '\n';
    }

    // code that doesn't need @fullName

    return 0;
}
```

`fullName` is accessible in the entire `if-statement`, including its bodies. *init-statements* were added to the language in C++17.

5.3 -- Switch statements

Index

5.1 -- Control flow introduction

C++ TUTORIAL | PRINT THIS POST

124 comments to 5.2 — If statements

« Older Comments 1 2

Jacob Salomon
January 8, 2020 at 5:40 pm · Reply

Hi nascardriver or Alex (HMMmm Has anyone ever seen you both together in the same room? :-))
Anyway, I wanted to make sure I undestood you correctly in your "other example", where you have a long chain of if/else if:

```
1   if (x < 0)
2       std::cout << x << " is negative\n";
3   else if (x < 10)
4       std::cout << x << " has 1 digit\n";
5   else if (x < 100)
6       std::cout << x << " has 2 digits\n";
7   else if (x < 1000)
8       std::cout << x << " has 3 digits\n";
9   else if (x < 10000)
10      std::cout << x << " has 4 digits\n";
11  else
12      std::cout << x << " was larger than 9999\n";
```

If I read correctly, these is a bunch of nested if's, as in:

```
1   if (x < 0)
2   {
3     std::cout << x << " is negative\n";
4     else
5     {
6       if (x < 10)
7       std::cout << x << " has 1 digit\n";
8       else
```

```
        {
          if (x < 100)
            std::cout << x << " has 2 digits\n";
          else
          {
            if (x < 1000)
              std::cout << x << " has 3 digits\n";
            else
            {
              if (x < 10000)
                std::cout << x << " has 4 digits\n";
              else
              {
                std::cout << x << " was larger than 9999\n";
              }
            }
          }
        }
      }
    }
```

(Cue: Fake Russian accent) I find that verrry difficult to believe, Mr. Smart. (Yeah, that reference definitely dates me! :-)

PLEASE tell me that's wrong! I don't recall getting this kind of warning when I learned C language.

Thanks!

nascardriver
January 9, 2020 at 4:04 am · Reply

Warning? That's just how it is, you don't have to think of it like that. There is no `else if` as such, it's just an `else` followed by another `if`. Right now, there's only one situation in which you have to remember this, `if constexpr`. `if constexpr` is an if-statement that is executed at compile-time, really useful when mixing types (Templates/auto is covered later).

```
if constexpr (x)
{
  // compile time
}
else if (y)
{
  // oops, run time
}

// correct:

if constexpr (x)
{
  // compile time
}
else if constexpr (y)
{
  // compile time
}
```

kitabski
December 10, 2019 at 8:00 am · Reply

Init-statements
I use CodeBlock and the init statement doesn't compile.
The error is:

C:\CBPprojects\LearncppUnit5\main.cpp|42|error: expected ')' before ';' token|

What might the problem?

```
#include <iostream>
#include <string>

int main(){

    using namespace std;

    string firstName{}, secondName{};

    cout << "Enter your first name, please: ";
    cin >> firstName;

    cout << "Enter your second name, please: ";
    cin >> secondName;

    if (string fullName {firstName + ' '+ secondName}; fullName.length() < 20 ){
        cout << fullName;
    }
    else
        cout << "Your name is too long";


    return 0;
}
```

nascardriver
December 11, 2019 at 5:08 am · Reply

init-statements were added in C++17. Make sure you selected that, or a higher one, in your project settings. I added a note to the lesson.

kitabski
December 11, 2019 at 6:06 am · Reply

Thank's for clarification!!!

omarbgm
November 1, 2019 at 5:34 am · Reply

#include <iostream>

using namespace std;

namespace Animals
{
enum Animals
{
Chicken,
dog,
cat,

```
    elephant,
    duck,
    snake,
    All_Animals
    };
}
```

Hey im a newbie at c++ and iwas wondering why when i excute this code what ever the number i write i get the out pur of chicken only

```
int main()
{
string Chicken;
string dog;
string snake;
string elephant;
string cat;
string duck;

int AnimalsLegs[Animals::All_Animals] = {2 ,4 ,4 ,4 ,2 ,0};
int input;
cout << "Choose an animal from these by writing their number : 1(Chicken) ; 2(Elephant) ; 3
cin >> input;
if (input = 1)
{
   cout << "Chicken got " << AnimalsLegs[Animals::Chicken] << "Legs";
}
else if (input == 2)
{
   cout << "elephant got " << AnimalsLegs[Animals::elephant] << "Legs";
}
else if (input == 3)
{
   cout << "dog got " << AnimalsLegs[Animals::dog] << "Legs";
}

else if (input == 4)
{
   cout << "cat got " << AnimalsLegs[Animals::cat] << "Legs";
}
else if (input == 5)
{
   cout << "duck got " << AnimalsLegs[Animals::duck] << "Legs";
}
else
{
   cout << "snake got " << AnimalsLegs[Animals::snake] << "Legs";
}
    return 0;
}
```

nascardriver
November 1, 2019 at 5:55 am · Reply

Line 14 is an assignment

sana ullah
October 24, 2019 at 8:05 am · Reply

how to console 1 to 100 in if statement.
and 1 to 100 negative.

Samira Ferdi
August 22, 2019 at 7:42 pm · Reply

Hi, Alex and Nascardriver!

I found others code like this:

```
int c{};
if(c = 10)
    std::cout << c; //print 10
```

I know how the code work, but what is your recommendation (because others do this)?

nascardriver
August 23, 2019 at 2:06 am · Reply

What is this supposed to solve or achieve?

```
std::cout << 10;
```

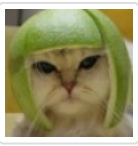
Parsa
July 27, 2019 at 6:27 pm · Reply

Visual Studio 2019 does not let me use an enum as a return type for a function(or at least I think that's what it's trying to tell me)

What the error says: missing type specifier-int assumed. Note: C++ does not support default-int

```
Check errorCheck(char oper)
{
    if (oper == '+')
        return Check::SUCCESS;
    else if (oper == '-')
        return Check::SUCCESS;
    else if (oper == '*')
        return Check::SUCCESS;
    else if (oper == '/')
        return Check::SUCCESS;
    else if (oper == '%')
        return Check::SUCCESS;
    else
        return Check::ERROR;
}
```

edit: I just realized I should have used a switch statement.



Alex
July 29, 2019 at 9:50 am · Reply

Could be a circular dependency issue. Is enum Check defined above this function (or in a separate header that is included?).

Can you post your full code?

Parsa
July 29, 2019 at 11:06 am · Reply

No, the enum wasn't, that was the issue. Thanks.

Samira Ferdi
July 4, 2019 at 7:43 pm · Reply

Hi, Alex and Nascardriver!

I'm just curious about if-else statement and I just want to make sure my understanding about if-else.

```
#include <iostream>

int main()
{
    std::cout << "Enter a positive number between 0 and 9999: ";
    int x;
    std::cin >> x;

    if (x < 0)
        std::cout << x << " is negative\n";
    else if (x < 10)
        std::cout << x << " has 1 digit\n";
    else if (x < 100)
        std::cout << x << " has 2 digits\n";
    else if (x < 1000)
        std::cout << x << " has 3 digits\n";
    else if (x < 10000)
        std::cout << x << " has 4 digits\n";
    else
        std::cout << x << " was larger than 9999\n";

    return 0;
}
```

In the code above, if I entered 9, then the result is "19 has 2 digits". This is true because 19 has digits. I Agree with that. It's true that 19 is less than 100 but it is true also that 19 is less than 1000, and less than 10000. Why the result just is "19 has 2 digits"? I think the result should be:
"19 has 2 digits"
"19 has 3 digits"
"19 has 4 digits"

nascardriver
July 5, 2019 at 12:41 am · Reply

You're using `else if`. Once 1 condition is true, all `else if`s after that are ignored.

Singh
June 11, 2019 at 10:30 am · Reply

I am newbee to C++, can you help me to simplify below program. It do revolve the problem but code is quite lenthy which I think can be done in less code lines.

```
//Program is to get the payment result based on user input.
//Such as 20% deduction till $500, 10% deduction till 10k & 5% deduction after words.
```

```
3
4   #include <iostream>
5
6   using namespace std;
7
8   int main()
9   {   int total; //variable initialization.
10
11      cout<<"Project is to calculate total amount to be paid and fees." <<endl;
12      cout<<"Please enter the total amount: "; cin>>total;
13
14      if(total <= 500) //will work only if for payment of upto $500
15      {
16          double fee{total*20/100}; //calculate total 20% fee.
17          cout<<"\n20% Fee of Total: " <<fee <<endl;
18          cout<<"Available Amount: " <<total - fee;
19      }
20          else if(total>500 && total<=10000)
21          {
22              double remain{total-500}; //total on which 10% will be deducted.
23              double fee{remain*10/100}; //fee on total after $500
24              double amount{total-fee-100}; //will deduct 20% till 500 and 10% after words.
25              cout<<"\nTotal Fee: " <<fee+100; //fee is 10% on after $500 and 100 is 20% of $
26              cout<<"\nAvailable Amount: " <<amount;
27          }
28              else
29              {
30                  double remain{total-10000}; //amount on which 5% need as fee
31                  double fee{remain*5/100}; //5% on amount after $10K
32                  double fee1{500*20/100}; //20% fee till $500
33                  double fee2{9500*10/100}; //10% fee till $10k
34
35                  cout<<"Total Fee is: " <<fee+fee1+fee2 <<endl;
36                  cout<<"Available Amount: " <<total-(fee+fee1+fee2) <<endl;
37              }
38      return 0;
39  }
```

nascardriver
June 12, 2019 at 2:56 am · Reply

- Limit your lines to 80 characters in length for better readability on small displays.
- Print the line feed at the end of the line, not at the beginning of the next line.
- Inconsistent formatting. Use your editors auto-formatting feature.
- Don't use "using namespace".
- Use double literals for doubles.

You can't reduce the line count, and doing so should not be a goal of yours. You could move the different fee calculations into functions, but they can't be merged, because they're too different from each other.
Line (17), 18, 25, 26, 35, 36 should be moved outside of the conditional blocks.

Singh
June 14, 2019 at 2:10 am · Reply

Thanks alot Alex, It will surely work as you have specified. Cheers!!

nascardriver

June 14, 2019 at 2:31 am · Reply

You're welcome! I'm not Alex though, Alex has a cat avatar and a blue message background :)

Singh

June 17, 2019 at 8:35 am · Reply

I am really Sorry, I thought of you as Alex. I think I should correct it now "Thank you NascarDriver" ;). I will get back as soon as I got a query.

Do you have Skype or anything other IM for direct communication as need some consultation regarding the carrier point of view. Thanks again, Cheers!!

Eric Mackay

April 12, 2019 at 3:58 pm · Reply

Alex,

Is there a good way to take a number inputted by the user and look up a result based on the number that is not a whole bunch of if / if else statements? As an example: in an RPG there might be a table listing numerous outcomes based on the random dice rolled by the player. Is there a way to take that random number and search for the corresponding answer without coding something like this:

```
1 if (userRoll == 1)
2   std::cout << "You hit your friend.";
3 else if (userRoll >=2 && userRoll <=24)
4   std::cout << "You dropped your sword.";
5 else if (userRoll == 25)
6   std::cout << "Your enemy pushed you over a cliff -- Bye, bye.";
```

I've been trying to code some useful tables for my gamer son using the above method and was thinking there's gotta be a better way. Some tables are 3 items long, others are 200+.

Any help would be appreciated.

Eric.

Alex

April 14, 2019 at 8:06 am · Reply

I can only think of 3 ways:
1) Use a series of if/else
2) Use a table to store the ranges and results and use a loop to do the iteration and comparison
3) Use a table lookup where you initialize an array with your results and the use the userRoll as the array index.

Eric Mackay

April 15, 2019 at 9:47 am · Reply

Thanks, Alex!

I was afraid there was no simple solution. I toyed with the idea of using a switch statement based on the dice roll. This might work for a situation where there is one result per number on the dice, but in my example above it would be silly and more cumbersome to use...

Thanks again for the ideas!

Eric.

nascardriver
April 15, 2019 at 10:56 am · Reply

Alex' second suggestion can be used, but you'll need arrays (Next chapter).

Matt
March 14, 2019 at 6:04 pm · Reply

Hi Alex!

I've been away for a while and am currently doing review of old lessons.

My question is about style/technique here, in the following code:

```
enum class ErrorCode
{
    ERROR_SUCCESS = 0,
    ERROR_NEGATIVE_NUMBER = -1
};
```

Since enum class is used instead of a standard enum, isn't it unnecessary to disambiguate the members with ERROR_... considering accessing them already requires using the ErrorCode:: prefix?

Thanks,
Matt

nascardriver
March 15, 2019 at 7:04 am · Reply

Hi Matt!

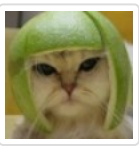
It is unnecessary.

nascardriver
December 1, 2018 at 3:22 am · Reply

Hi Alex!

This lesson is missing the optional init-statement

```
if (std::size_t sFound{ strURL.find("://") }; sFound != std::string::npos)
{
  // ...
}
```



Alex
December 2, 2018 at 3:54 pm · Reply

Yup, that was added in C++17 and the lesson hasn't been updated yet. I'll add that to my notes and make sure it gets included in the update.

Nguyen
October 22, 2018 at 9:36 pm · Reply

Hi nascardriver,

I need your help. Is there any Date type? I just came up with a simple and small program.
Here is my example: if the due date is before 10/25/2018, then you will not be fined. If the due date is between 10/26/2018 and 11/26/2018, then you will be fined $100. If the due date is after 11/26/2018, then you will be fined $200.

Here is my pseudocode:

```
date dueDate;

std::cout << "Enter the due date: ";
std::cin >> dueDate;
if (dueDate > 10/25/2018)
   std::cout << "No fine";
else if (dueDate < 10/25/2018 && dueDate > 11/26/2018)
   std::cout << "Fine: $100";
else
   std::cout << "Fine: $200";
```

Could you please help me to code my example?
I'd like to know if I set the conditions correctly.

Thanks, Have a great day.

Alex
October 23, 2018 at 8:09 am · Reply

https://en.cppreference.com/w/cpp/chrono is the C++ way of handling time and dates. Personally I haven't had the chance to use the date functionality here, but it would make a good future lesson. I'll add it to my to-do list.

nascardriver
October 23, 2018 at 11:00 am · Reply

Hi Nguyen!

As Alex said, @std::chrono is the way to go. Date and Time are terrible to work with, because there's no real logic behind. A minute is 60 seconds, a day is 24 hours, a month is 28-31 depending on the year and so on. If you can't solve it with @std::chrono, your best bet is to convert everything to seconds and go from there.

Juan
July 19, 2018 at 3:38 am · Reply

How can I initialise a constant inside an if statement? Something like

```
int main ()
{
  int input;
  std::cin >> input;

  if (input == 1)
    const int a = 1;
  else if (input == 2)
    const int a = 2;
  else
    const int a = 3;
```

```
12
13      std::cout << a << std::endl;
14
15      return 0;
16  }
```

I either initialise it outside the if and it cannot be const, or I initialise it inside and I cannot use it outside the if.

Thanks

nascardriver
July 19, 2018 at 3:44 am · Reply

Hi Juan!

```
1   // Using the conditional operator
2   const int a{ (input == 1) ? 1 : (input == 2) ? 2 : 3 };
3
4   // Or using a function
5   int getValueForConst(int iInput)
6   {
7     if (input == 1)
8       return 1;
9     // ...
10  }
11
12  const int a{ getValueForConst(input) };
13
14  // Or using a lambda
15  const int a{ [](int iInput)
16    {
17      if (iInput == 1)
18        return 1;
19      // ...
20    }(input)
21  };
```

References
Lesson 3.4 - Sizeof, comma, and conditional operators

Juan
July 19, 2018 at 3:47 am · Reply

ok, great, I was hoping I could avoid conditional operators, but I'll have to settle for one of my options. Thanks!

nascardriver
July 19, 2018 at 3:48 am · Reply

You could also create a non-const variable first, assign the value with a normal if-statement and initialize a const with that variable. I can't think of a reason why you would need a const variable though.

Benjamin
April 24, 2018 at 7:20 pm · Reply

In this example below you have:

```
std::cout << x << "is between 10 and 20 (inclusive)\n";
```

Shouldn't that be "exclusive" not "inclusive"?

Also

```
std::cout << x << "is less than 10\n";
```

should say "is 10 or less"

```
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int x;
    std::cin >> x;

    if (x > 10)
    {
        if (x < 20)
            std::cout << x << "is between 10 and 20 (inclusive)\n";
    }
    else // attached to outer if statement
        std::cout << x << "is less than 10\n";

    return 0;
}
```

Alex
April 29, 2018 at 10:33 am · Reply

Yes, quite right. I've updated the example a bit so that the numbers are now actually between 10 and 20 inclusive in this case.

Bud Roszales
March 31, 2018 at 8:44 am · Reply

This website is completely gret. I've researched these details a great deal and
I view it that is good written, fast to comprehend.
I congratulate you because of this research that I am going to recommend to the people around.
I ask you to recommend the gpa-calculator.co site where each university student or
learner can find results grade point average rating.

Thank you!

Ali Dahud
March 15, 2018 at 5:58 am · Reply

```
ErrorCode doSomething(int value)
{
    // if value is a negative number
    if (value < 0)
```

```
        // early return an error code
        return ErrorCode::ERROR_NEGATIVE_NUMBER;

    // Do whatever here

    return ErrorCode::ERROR_SUCCESS;
}

int main()
{
    std::cout << "Enter a positive number: ";
    int x;
    std::cin >> x;

    if (doSomething(x) == ErrorCode::ERROR_NEGATIVE_NUMBER)
    {
        std::cout << "You entered a negative number!\n";
    }
    else
    {
        std::cout << "It worked!\n";
    }

    return 0;
}
```

Could anyone please explain me this code?
in Detai please

nascardriver
March 15, 2018 at 8:07 am · Reply

Hi Ali!

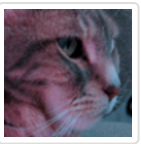
If you want a detailed description you should provide full code.

```
#include <iostream>

enum class ErrorCode
{
  ERROR_NEGATIVE_NUMBER,
  ERROR_SUCCESS
};

// Create a new function called 'doSomething' that returns and @ErrorCode and
// takes an int as parameter.
ErrorCode doSomething(int value)
{
  // If the value is negative
  if (value < 0)
  {
    // Return ErrorCode::ERROR_NEGATIVE_NUMBER to signal the caller that the
    // value they passed is negative. A negative value would probably cause this
    // function to crash or not work properly.
    return ErrorCode::ERROR_NEGATIVE_NUMBER;
  }

  // Code that depends an @value being greater than or equal to 0


  // Return ErrorCode::ERROR_SUCCESS to signal the caller that everything worked
  // as expected.
```

```
27    return ErrorCode::ERROR_SUCCESS;
28  }
29
30  int main()
31  {
32    // Print "Enter a positive number: " to the console
33    std::cout << "Enter a positive number: ";
34    // Create an int with value 0
35    int x{ 0 };
36    // Read input from the console and store it in @x
37    std::cin >> x;
38
39    // Call @doSomething with argument @x and compare the return value of
40    // @doSomething to ErrorCode::ERROR_NEGATIVE_NUMBER
41    if (doSomething(x) == ErrorCode::ERROR_NEGATIVE_NUMBER)
42    {
43      // Print to console
44      std::cout << "You entered a negative number!\n";
45    }
46    else
47    {
48      // Print to console
49      std::cout << "It worked!\n";
50    }
51
52    // Return 0 to signal successful program execution
53    return 0;
54  }
```

Ali Dahud
March 16, 2018 at 1:34 am · Reply

now it's clear thank you :)



Peter Baum
March 12, 2018 at 5:59 pm · Reply

Under chaining if statements - I felt confused encountering “else if” because it wasn’t really explained in detail. When I saw the second “else” in the example, I didn’t know if the “else” functioned relative to the “else if” or the “if” at the top.

Perhaps just adding a little more detail to this section will help.

Alex
March 13, 2018 at 2:05 pm · Reply

Done. Thanks for the feedback!

Andrew
January 11, 2018 at 9:30 am · Reply

What is the goal of using "null statement"?

Alex
January 11, 2018 at 1:54 pm · Reply

Null statements are typically used when the language requires a statement to exist but the programmer doesn't need one. Most often, this occurs in conjunction with loops, where the termination condition of the loop is performing all of the evaluation necessary.

For example:

```
while ( playGame() )
    ; // do nothing
```

This code will continue to execute playGame() as long as playGame() returns boolean true.

Peter Baum
March 12, 2018 at 5:58 pm · Reply

I would like to see this moved into the lesson. Could you also provide another example? Personally I learn better when I am shown how I might use something rather than trying to retain it in my memory for some unknown purpose in the future.

Alex
March 13, 2018 at 2:05 pm · Reply

I updated the text a bit -- the challenge here is that null statements are almost never used with if statements, so any example would be equally contrived. This is here more to serve as a warning of what to watch out for, than to be something I expect you to use yet.

Thanks for the feedback.

IMRAN ALI
December 10, 2017 at 2:50 am · Reply

Is it possible for us to create square root function by ourselves with the knowledge we had up to this point?

Jozef
December 11, 2017 at 10:02 am · Reply

I think you should be able to. But Im no expert just learning as you.

Alex
December 12, 2017 at 3:27 pm · Reply

Up to this lesson? Probably not. But once you finish this chapter, then you should be able to use an iterative method (via a loop) for calculating a square root.

April
December 5, 2017 at 8:44 pm · Reply

I'm still having a hard time understanding the benefit of using enumerated value error codes. Would you mind elaborating, please?

Alex
December 7, 2017 at 9:57 pm · Reply

It's easier to understand what ERROR_CANT_OPEN_FILE means than some magic number return value, like -2.

Kushagra
October 11, 2017 at 2:33 am · Reply

In most of the cases return value of int main()
is 0. Then why don't we use void main() instead of int main()? Please explain.

Alex
October 12, 2017 at 3:45 pm · Reply

Because functions defined with a void return type can't return a value at all (not even a 0).

mark
November 27, 2017 at 1:10 pm · Reply

Alex you do have this issue in the example above where you explain implicit blocks void main()+return 0

```
1 void main()
2
3 // code lines
4
5 return 0;
```

Alex
November 29, 2017 at 4:30 pm · Reply

Oops. Changed all the void main() to int main() as they should be.

Gurdeep Singh
September 19, 2017 at 5:59 pm · Reply

Just asking out of curiosity
Are there any plans to make tutorials for other programming languages.

Alex
September 21, 2017 at 3:48 pm · Reply

Nope.

« Older Comments 1 2