4.10 — Introduction to if statements

BY ALEX ON APRIL 23RD, 2019 | LAST MODIFIED BY NASCARDRIVER ON NOVEMBER 12TH, 2019

Consider a case where you're going to go to the market, and your roommate tells you, "if they have strawberries on sale, buy some". This is a conditional statement, meaning that you'll execute some action ("buy some") only if the condition ("they have strawberries on sale") is true.

Such conditions are common in programming, as they allow us to implement conditional behavior into our programs. The simplest kind of of conditional statement in C++ is called an *if statement*. An **if statement** allows us to execute one (or more) lines of code only if some condition is true.

The simplest *if statement* takes the following form:

```
if (condition) statement;
```

For readability, this is more often written as following:

```
if (condition)
    statement;
```

A **condition** (also called a **conditional expression**) is an expression that evaluates to a Boolean value.

If the *condition* of an *if statement* evaluates to Boolean value *true*, then the subsequent *statement* is executed. If the *condition* instead evaluates to Boolean value *false*, the subsequent *statement* is skipped.

A sample program using an if statement

Given the following program:

```
#include <iostream>

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

    if (x == 0)
        std::cout << "The value is zero\n";

    return 0;
}
```

Here's output from one run of this program:

```
Enter an integer: 0
The value is zero
```

Let's examine how this works in more detail.

First, the user enters an integer. Then the condition *x == 0* is evaluated. The *equality operator* (==) is used to test whether two values are equal. Operator== returns *true* if the operands are equal, and *false* if they are not. Since *x* has value 0, and *0 == 0* is true, this expression evaluates to *true*.

Because the condition has evaluated to *true*, the subsequent statement executes, printing *You entered zero*.

Here's another run of this program:

```
Enter an integer: 5
```

In this case, *x == 0* evaluates to *false*. The subsequent statement is skipped, the program ends, and nothing else is printed.

Warning

If statements only conditionally execute a single statement. We talk about how to conditionally execute multiple statements in lesson **5.2 -- If statements**.

If-else

Given the above example, what if we wanted to tell the user that the number they entered was non-zero?

We could write something like this:

```
#include <iostream>

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

    if (x == 0)
        std::cout << "The value is zero\n";
    if (x != 0)
        std::cout << "The value is non-zero\n";

    return 0;
}
```

Or this:

```
#include <iostream>

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

    bool zero { (x == 0) };
    if (zero)
        std::cout << "The value is zero\n";
    if (!zero)
        std::cout << "The value is non-zero\n";

    return 0;
}
```

Both of these programs are more complex than they need to be. Instead, we can use an alternative form of the *if statement* called *if-else*. *If-else* takes the following form:

```
if (condition)
    true_statement;
else
    false_statement;
```

If the *condition* evaluates to Boolean true, *true_statement* executes. Otherwise *false_statement* executes.

Let's amend our previous program to use an *if-else*.

```
#include <iostream>

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

    if (x == 0)
        std::cout << "The value is zero\n";
    else
        std::cout << "The value is non-zero\n";

    return 0;
}
```

Now our program will produce the following output:

```
Enter an integer: 0
The value is zero


Enter an integer: 5
The value is non-zero
```

Chaining if statements

Sometimes we want to check if several things are true or false in sequence. We can do so by chaining an *if statement* to a prior *if-else*, like so:

```
#include <iostream>

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

    if (x > 0)
        std::cout << "The value is positive\n";
    else if (x < 0)
        std::cout << "The value is negative\n";
    else
        std::cout << "The value is zero\n";

    return 0;
}
```

The *less than operator* (<) is used to test whether one value is less than another. Similarly, the *greater than operator* (>) is used to test whether one value is greater than another. These operators both return Boolean values.

Here's output from a few runs of this program:

```
Enter an integer: 4
The value is positive


Enter an integer: -3
The value is negative


Enter an integer: 0
The value is zero
```

Note that you can chain *if statements* as many times as you have conditions you want to evaluate. We'll see an example in the quiz where this is useful.

Boolean return values and if statements

In the previous lesson (**4.9 -- Boolean values**), we wrote this program using a function that returns a Boolean value:

```
#include <iostream>

// returns true if x and y are equal, false otherwise
bool isEqual(int x, int y)
{
    return (x == y); // operator== returns true if x equals y, and false otherwise
}

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

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

    std::cout << std::boolalpha; // print bools as true or false

    std::cout << x << " and " << y << " are equal? ";
    std::cout << isEqual(x, y); // will return true or false

    return 0;
}
```

Let's improve this program using an *if statement*:

```
#include <iostream>

// returns true if x and y are equal, false otherwise
bool isEqual(int x, int y)
{
    return (x == y); // operator== returns true if x equals y, and false otherwise
}
```

```
8
9   int main()
10  {
11      std::cout << "Enter an integer: ";
12      int x {};
13      std::cin >> x;
14
15      std::cout << "Enter another integer: ";
16      int y {};
17      std::cin >> y;
18
19      std::cout << std::boolalpha; // print bools as true or false
20
21      if (isEqual(x, y))
22          std::cout << x << " and " << y << " are equal\n";
23      else
24          std::cout << x << " and " << y << " are not equal\n";
25
26      return 0;
27  }
```

Two runs of this program:

```
Enter an integer: 5
Enter another integer: 5
5 and 5 are equal


Enter an integer: 6
Enter another integer: 4
6 and 4 are not equal
```

In this case, our conditional expression is simply a function call to function *isEqual*, which returns a Boolean value.

Non-Boolean conditionals

In all of the examples above, our conditionals have been either Boolean values (true or false), Boolean variables, or functions that return a Boolean value. What happens if your conditional is an expression that does not evaluate to a Boolean value?

In such a case, the conditional expression is converted to a Boolean value: non-zero values get converted to Boolean *true*, and zero-values get converted to Boolean *false*.

Therefore, if we do something like this:

```
1   #include <iostream>
2
3   int main()
4   {
5       if (4) // nonsensical, but for the sake of example...
6           std::cout << "hi";
7       else
8           std::cout << "bye";
9
10      return 0;
11  }
```

This will print “hi”, since 4 is a non-zero value that gets converted to Boolean *true*, causing the statement attached to the *if* to execute.

We'll continue our exploration of *if statements* in future lesson **5.2 -- If statements**.

Quiz time

Question #1

A prime number is a whole number greater than 1 that can only be divided evenly by 1 and itself. Write a program that asks the user to enter a single digit integer. If the user enters a single digit that is prime (2, 3, 5, or 7), print "The digit is prime". Otherwise, print "The digit is not prime".

Show Hint

Show Solution

 **4.11 -- Chars**

 **Index**

 **4.9 -- Boolean values**

C++ TUTORIAL | PRINT THIS POST

45 comments to 4.10 — Introduction to if statements

Tom Bauer
January 27, 2020 at 12:24 pm · Reply

I just wanted to know what you think about this solution.

[code]

'Quelle.cpp'

```
#include <iostream>
#define PEAN "Please enter a number: "

int duoble(int x)
{
   return x * 2;
}

int multiply(int x,int y)
{
   return x * y;
}

int subtract(int x,int y)
{
   return x - y;
}

int divide(int x,int y)
{
   return x/y;
}

int add(int x, int y)
{
   return x + y;
}

int userinput()
{
   std::cout << PEAN << '\n';
   int input{};
   std::cin >> input;
   return input;
}

int userinput2()
{
   std::cout << "Please enter another number: "<< '\n';
   int input{};
   std::cin >> input;
   return input;
}
```

'main.cpp'

```
#include "Host.h"

void bothnegative()
{
   std::cout << "Both numbers are negative." << std::endl;
}

void twobiggerzero()
{
```

```
    std::cout << "Your second number is positive." << std::endl;
}

void twolesserzero()
{
    std::cout << "Your second number is negative." << std::endl;
}

void onebiggerzero()
{
    std::cout << "Your first number is positive." << std::endl;
}

void onelesserzero()
{
    std::cout << "Your first number is negative." << std::endl;
}

void onebiggertwo()
{
    std::cout << "Your first number is greater than your second." << std::endl;
}

void twobiggerone()
{
    std::cout << "Your second number is greater than your first." << std::endl;
}

void notsamenum()
{
    std::cout << "You entered two different numbers!" << std::endl;
}

void samenum()
{
    std::cout << "You entered the same number!" << std::endl;
}

bool boolerx(int x)
{
    return (x < 0);
}

bool boolery(int y)
{
    return (y < 0);
}

void boolertestnegative(int x)
{
    if (x == 1) std::cout << "Both numbers are negative." << std::endl;
}

void boolertestpositiv(int x)
{
    if (x == 0) std::cout << "Both numbers are positive." << std::endl;
}

int main()
{
    int x = userinput();
```

```
    int y = userinput2();
    if (x == y) samenum();
    else notsamenum();
    if (x < y) twobiggerone();
    if (x > y) onebiggertwo();

    if (boolerx(x) == boolery(y)) boolertestnegative(boolerx(x));
    else if (x < 0) onelesserzero();
    else if (y < 0) twolesserzero();
    if (boolerx(x) == boolery(y)) boolertestpositiv(boolerx(x));
    else if (x > 0) onebiggerzero();
    else if (y > 0) twobiggerzero();
}
```

[code]

nascardriver
January 28, 2020 at 2:11 am · Reply

Closing code tags use a forward slash [/code].

- Avoid preprocessor macros. They can break code and are harder to understand than C++.
- Initialize variable with brace initialization for higher type safety.
- Use '\n' unless you need `std::endl`, it's faster.
- `boolerx` and `boolery` are identical. Variable names don't matter.
- Avoid abbreviations.
- You names are difficult to read. Consider camel case or underscores.

Tony98
January 25, 2020 at 1:12 pm · Reply

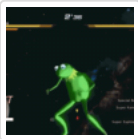
Hi (again), I've done the exercise in a different way since I know about operator "||" (I've previously done this tutorial but came back since I had problems and needed a fresh restart). Since I couldn't do it like you did I've done it in a different way. Could you please tell me if everything's alright or whether I should have proceeded in a different way? Thank you for the hard work!

```
1  int userInput()
2  {
3      std::cout << "Please enter a single digit integer: ";
4      int x{};
5      std::cin >> x;
6      return x;
7  }
8
9  bool isPrime(int x)
10 {
11     return (x==2 || x==3 || x==5 || x==7);  // returns true if the user's value is 2, 3, 5
12 }
13
14 int main()
15 {
16     int userValue{ userInput() };
17     if (isPrime(userValue))
18         std::cout << "The digit is prime";
19     else
20         std::cout << "The digit is not prime";
21
22     return 0;
```

```
}
```

nascardriver
January 26, 2020 at 1:37 am · Reply

Your code is solid. If you know something, you can use it.



Kermit
January 17, 2020 at 10:32 am · Reply

here is quiz answer

```
#include <iostream>

using namespace std;
/*
 Write a program that asks the user to enter a single digit integer.
 If the user enters a single digit that is prime (2, 3, 5, or 7), print “The digit is prime
 Otherwise, print “The digit is not prime”.
*/

int userInput()
{
    cout << "Enter a single digit integer: ";
    int x{};
    cin >> x;
    return x;
}
void printIsPrime(int x)
{
    string x1{ "The digit is prime.\n\n" };
    if (x == 2)
    {
        cout << x1;
    }
    else if (x == 3)
    {
        cout << x1;
    }
    else if (x == 5)
    {
        cout << x1;
    }
    else if (x == 7)
    {
        cout << x1;
    }
    else
    {
        cout << "The digit is not prime.\n\n";
    }
}
int main()
{
    while (true)
    {
        int x{ userInput() };
        printIsPrime(x);
    }
```

```
48      return 0;
49  }
```

nascardriver
January 18, 2020 at 1:19 am · Reply

Good job!

By merging `print` and `isPrime`, you're decreasing reusability, ie. you can no longer check if a number is prime without printing it. Have a look at the solution in case you didn't understand the return.

Kermit
January 18, 2020 at 2:18 am · Reply

that's correct. understood thanks

BooGDaaN
November 12, 2019 at 6:23 am · Reply

In section if-else, you need to update the code. At line 9

```
1  bool zero { (x == 0) }
```

is missing ';'.

nascardriver
November 12, 2019 at 6:33 am · Reply

Updated, thanks!

Gustav
November 8, 2019 at 1:16 am · Reply

Hi Alex!

Just wondering if this code is acceptable

Inside io.cpp

```
1  bool isPrime(int x){
2
3      int p = x % 2;
4
5      if(x == 2){ // explicitly saying that number 2 is a prime
6          return true;
7      }
8      else if (p == 0){ // divisble by 2 therefore it's not a prime
9          return false;
10     }
11     else{
12         return true;
13     }
14 }
```

Fellow traveller

November 8, 2019 at 5:10 pm · Reply

Good try but nope. This would tell you that 9 is prime because it is neither 2 nor divisible by two without remainder.

Gustav

November 8, 2019 at 10:01 pm · Reply

How about this one?

```
1  bool isPrime(int x){
2      int n;
3      for(int i = 2; i <= x/2; i++){
4          n = x % i;
5          if(n == 0){ // Divisible therefore a prime
6              return false;
7          }
8      }
9
10     return true;
11 }
```

~Cheers!

nascardriver

November 9, 2019 at 3:04 am · Reply

`n` should be declared inside of the loop. Initialize your variables with brace initializers. Use ++prefix unless you need postfix++.

You code now works, but it's slow. An easy optimization is to check for divisibility by 2 first, then use `i += 2` instead of `++i`. Every number that is divisible by an even number, would have been divisible by 2 already, so line 5 will never be true for anything except `i == 2`.

Gustav

November 9, 2019 at 7:58 pm · Reply

quick question, does it check for other numbers as well (numbers that divisible by 3 or any odd numbers)?

nascardriver

November 10, 2019 at 1:58 am · Reply

The code you posted checks for divisibility by all numbers from 2 up to x/2.

Gustav

November 10, 2019 at 8:41 am · Reply

```
1  bool isPrime(int x){
2      for(int i = 2; i <= x/2; i+=2){
3          int n{x % i};
4          if(n == 0){ // Divisible therefore a prime
```

```
            return false;
        }
    }

    return true;
}
```

Something like that?
~Cheers!

nascardriver
November 11, 2019 at 4:08 am · Reply

No, that won't work, you should've noticed that while testing. When you initialize `i` to 2, and do `i += 2`, you're only testing for divisibility by even numbers.
`i` should be initialized with brace initialization to 3, check divisibility by 2 manually before the loop starts.
You can use another loop to test your code

```
for (int i{ 0 }; i < 1000; ++i)
{
  std::cout << i << ": " << isPrime(x) << '\n';
}
```

Gustav
November 12, 2019 at 12:13 am · Reply

Aha, I get it now.
Thanks!

~Cheers!

Adam
October 20, 2019 at 9:14 am · Reply

Can somebody please tell me why I'm getting told a ; is missing?

```
#include <iostream>

int main() {

    std::cout << "Please enter a number: ";
    int number{};
    std::cin >> number;

    if (number == 0) {
        std::cout << "The number is 0!";

    }
    else {
        std::cout << "The number you entered is not 0... I am most disappointed in you.";
    }

    std::cout << "Please enter another number: ";
    int number2{ 0 };
    std::cin >> number2;
```

```
    if (number2 == 0) {

        std::cout << "The number you entered is 0.";

    }
    else if (number2 > 0) {

        std::cout << "The number you entered is more than 0!";

    }
    else(number2 < 0) {

        std::cout << "The number you entered is less than 0!";

    }

}
```

nascardriver
October 20, 2019 at 9:17 am · Reply

Please use code tags and post the full error message.

```
1 else(number2 < 0) {
2 // elses don't have conditions. should be
3 else {
```

Sinethemba
August 14, 2019 at 6:24 am · Reply

I added a check to see if the number entered by the user was indeed a single digit. Is this fine?

```
1  #include <iostream>
2
3  // gets user input
4  int getUserInput(){
5
6      std::cout << "Enter a single digit integer: ";
7      int userValue {0};
8      std::cin>> userValue;
9
10     return userValue; // returns the value entered by the user
11 }
12
13 // Checks if the single digit is prime
14 void checkIfPrimeNumber(int userInput){
15     if(userInput == 2)
16         std::cout << "The digit is prime\n";
17     else if(userInput == 3)
18         std::cout << "The digit is prime\n";
19     else if(userInput == 5)
20         std::cout << "The digit is prime\n";
21     else if(userInput == 7)
22         std::cout << "The digit is prime\n";
23     else
24         std::cout << "The digit is not prime\n";
25 }
26
27 int main(){
28
29     int userInput {getUserInput()}; // get user input
```

```
30
31      while(userInput > 9){   //checks if the number entered by user is a single digit
32
33          std::cout << "The entered integer is not a single digit value!\n";
34          userInput = getUserInput();
35      }
36      // calling checkIfPrimeNumber() which validates if user input is prime
37      checkIfPrimeNumber(userInput);
38
39      return 0;
40  }
```

nascardriver
August 14, 2019 at 7:16 am · Reply

Yep

Jose Chavez
September 8, 2019 at 3:23 pm · Reply

that's a good one, I did the same thing, I applied the python stuff although I really liked the authors' answer a lot since, it's quite simple.

but clearly the difference of how they teach here btw those YouTube channels is very obvious, here things are much more complex and simple to understand. I don't know how to express my gratitude to these people

BlueManedHawk
July 26, 2019 at 10:51 am · Reply

I don't understand why we need to create a whole separate function in the quiz. The following code works fine, uses less lines, and in my opinion is easier to read. Obviously, if you were checking for single-digit primes multiple times in the same program, you would want to use a function, but it just seems so unneccessary.

```
1   #include <iostream>
2
3   int main ( ) {
4       std::cout << "Enter a single-digit integer: " ;
5       int input { } ;
6       std::cin  >> input ;
7
8       if ( input == 2 || input == 3 || input == 5 || input == 7 )
9           std::cout << "The number is prime.\n" ;
10      else
11          std::cout << "The number is not prime."
12  }
```

nascardriver
July 27, 2019 at 11:34 pm · Reply

Separating programs into smaller pieces increases reusability and in most cases readability. If you wanted to check if another number is prime, you'd have to repeat line 8, which is asking for trouble.

- `main`: Missing return-statement.
- If your program prints anything, the last thing it prints should be a line feed ('\n').

Parsa
July 17, 2019 at 8:26 am · Reply

What if you want the function isPrime to return something so that the (for example) third else if executes?

if you want the function to catch an error (for example you enter a number that isn't greater than one) and it returns something so that the third else if (or something) executes letting you know that the number you entered isn't greater than one?

Thanks.

Great site by the way.

nascardriver
July 17, 2019 at 8:50 am · Reply

You haven't learned this yet

```
#include <optional>

std::optional<bool> isPrime(int i)
{
  if (i <= 0)
  {
    return {};
  }
  else
  {
    // your code that checks if @i is prime
    return false;
  }
}

// ...

if (auto b{ isPrime(123) })
{
  // @isPrime returned false/true
  if (*b)
  {
    std::cout << "prime\n";
  }
  else
  {
    std::cout << "not prime\n";
  }
}
else
{
  // @isPrime failed
}
```

Parsa
July 17, 2019 at 11:18 am · Reply

Seems pretty confusing for now, but it's good to know that we are going to learn it later. Thank you.

nascardriver
July 18, 2019 at 12:01 am · Reply

`std::optional` isn't taught on learncpp. You'll learn about other ways of achieving the same and you'll learn everything required to understand `std::optional`.

David
July 3, 2019 at 2:19 am · Reply

Hi. What is the difference between declaring a variable using:
int x;

.. and declaring it with curly braces:
int x{} ?

and what is the advantage of using the braces, since both ways appear to have the same effect?

nascardriver
July 3, 2019 at 2:50 am · Reply

```
1 int x;
```

The value of `x` is undefined. Reading from `x` produces undefined behavior.

```
1 int x{};
```

`x`'s value is 0. Empty curly brackets initialize variables to their type's default value.

See lesson 1.4 and 1.6 for more information.

Neon
June 29, 2019 at 8:35 am · Reply

1) In the slightly more complicated example, there is a trailing << operator in line 10.

2) Would you not consider it a best practice to always use curly braces for if/else statement bodies? If not, why? I usually find them very clarifying. For one-liners I use them on the same line, though.

```
1 if (foo)
2 { bar(); }
```

Alex
June 30, 2019 at 5:35 pm · Reply

1) Typo fixed. Thanks!
2) A lot of people advocate for always using braces because it makes your code more resistant to breaking if additional statements are added or removed. That's a valid viewpoint. However, plenty of other programmers would argue that doing so makes your code harder to read or more

complicated than it needs to be. And since you can't assume braces exist, you need to check for them anyway. I think it's safer to do so than not, but personally I like to favor simplicity in this case.

alfonso
June 1, 2019 at 12:59 am · Reply

Hi again! You did not teach "else if" but your solution to the quiz uses it. I tried to use only what appeared in the course by now in my solution.

Alex
June 1, 2019 at 10:38 am · Reply

I've added a section about chaining if-statements together. Thanks for the feedback!

2+2=5
May 19, 2019 at 11:33 am · Reply

Executing multiple statements
U forgot { after else in the example line 10

Alex
May 20, 2019 at 10:17 am · Reply

It doesn't need it, since the statement that executes after the else is a single statement. I've updated the lesson to try to make this a bit more clear.

Jonathan
May 18, 2019 at 7:03 am · Reply

I am new to C++ and I thought the question asked me to test all possible input so I wrote the following program lol. But I don't understand why when I "run without debugging", the program stops if my input is very large, and can anyone tell whether my code is correct or not??

```
1  #include <iostream>
2  #include "Header.h"
3
4  int main()
5  {
6      int p = inputnum();
7      check1(p);
8
9      return 0;
10 }
11
12 int inputnum()
13 {
14     std::cout << "Enter a positive integer: ";
15     int num{};//ask for input
16     std::cin >> num;
17     return num;
18 }
19
20 void check1(int p)
21 {
```

```
    if (p == 1)//make sure the input is not 1
    print1();

    else
    check2(p, 2);//start testing with number 2
}

void check2(int pp, int div)
{
    int a = pp / div;//this fraction does not contain decimal
    double pp_ = pp;
    double div_ = div;
    double b{ pp_/ div_ };//double such that the fraction contains decimal(remainder if exi
    double c{ b - a };
    std::cout << c<<'\n';//observe the program running

    if(c==0)// if difference is 0, no remainder, hence divisible
    printnp(pp, div);

    else//if difference is not 0, the "div" is not a divisor of the input
    check3(pp, div);
}

void check3(int pp, int div)
{
    if (div > sqrt(pp))    // if div>sqrt(pp), we are sure that it is a prime number
                        // assume it is divisble, the quotient is smaller than "div"
                        // but this is tested before, and we know it is not a divisor
                        // which leads to a contradiction
    printp(pp);

    else
    {
        if (div == 2)
            div = div + 1;//shift 2 to 3
        else
            div = div + 2;//skip the even number as they must not be a divisor
    check2(pp, div);//back to check2() to use another divisor
    }
}

void print1()
{
std::cout << "1 is not a prime number.";
}

void printnp(int ppp, int div)
{
std::cout << ppp << " is not a prime number, it is divisible by "<<div<<".";
}// e.g. 4 is not a prime number, it is divisible by 2.

void printp(int ppp)
{
std::cout << ppp << " is a prime number.";
}// e.g. 17 is a prime number.
```

nascardriver
May 18, 2019 at 7:29 am · Reply

* Initialize your variables with brace initializers.
* Limit your lines to 80 characters in length for better readability on small displays.

* There's no need for a header, move the function definitions above @main.
* Use the auto-formatting feature of your editor.
* Name your variables descriptively.
* Use @std::sqrt instead of @sqrt.
* Missing include to <cmath> (or <math.h>).
* Floating point operations are prone to errors (ie. the result is not precise). This will cause your program to yield wrong results sooner or later. Use modulus instead.
* Missing printed trailing line feed (Print a line feed when your program is done).
* Use double literals for doubles (0.0 instead of 0).

> the program stops if my input is very large
Try changing the '\n' in line 36 to @std::endl. I'm guessing that you're stuck in an infinite loop and the output doesn't get flushed. @std::endl prints the text immediately, whereas the text might stay in memory for a while before it gets printed when using '\n'.

Jonathan
May 18, 2019 at 8:49 am · Reply

thank you so much!

nascardriver
May 18, 2019 at 8:50 am · Reply

You're welcome, let me know how it went!

Jonathan
May 18, 2019 at 8:09 pm · Reply

This is my updated code base on your suggestion! Some of your suggestions I not quite understand (1) what is a trailing line feed? (2) what is double literal? Is it different from double?

(3) One problem is that the program still stops in the middle when my input is large, for example using the number 2147483077:
When 2147483077 is divided by 4345 the remainder is 1587
When 2147483077 is divided by 4347 the remainder is 4219
When 2147483077 is divided by 4349 the remainder is 3414
When 2147483077 is divided by 4351 the remainder is 3517
When 2147483077 is divided by 4353 the remainder is 175
When
C:\Users\26jon\source\repos\Project1\Debug\Project1.exe (process 5792) exited with code -1073741571.
Press any key to close this window . . .
^^^The program stops at "When"^^^

```
1  #include <iostream>
2  #include <math.h>
3
4  int inputnum();
5  void print1();
6  void printnp(int num, int div);
7  void printp(int num);
8  void check1(int num);
9  void checkRemainder(int num, int div);
10 void changeDiv(int num, int div);
```

```
11
12  int main()
13  {
14      int num = inputnum();
15      check1(num);
16
17      return 0;
18  }
19
20  int inputnum()
21  {
22      std::cout << "Enter a positive integer: ";
23      int num{};//ask for input
24      std::cin >> num;
25      return num;
26  }
27
28  void check1(int num)
29  {
30      if (num == 1)//make sure the input is not 1
31      print1();
32
33      else
34      checkRemainder(num, 2);//start testing with number 2
35  }
36
37  void checkRemainder(int num, int div)
38  {
39      int remainder{ num % div };
40      std::cout << "When "<<num<<" is divided by "<<div<<", the remainder i
41
42      if (remainder == 0)//check for remainder
43          printnp(num, div);
44
45      else//if there is remainder 0, the "div" is not a divisor of the inpu
46          changeDiv(num, div);
47  }
48
49  void changeDiv(int num, int div)
50  {
51      if (div > std::sqrt(num))// if "div" > square root of the input
52          printp(num);        // then it is a prime number
53
54      else
55      {
56              if (div == 2)
57                  div = div + 1;//shift 2 to 3
58
59              else
60                  div = div + 2;//skip the even number as they must not be
61
62          checkRemainder(num, div);//back to check2() to use another divisc
63      }
64  }
65
66  void print1()
67  {
68      std::cout << "1 is not a prime number.";
69  }
70
71  void printnp(int nPrime, int div)
72  {
```

```
73      std::cout << nPrime << " is not a prime number, it is divisible by "<
74  }
75
76  void printp(int prime)
77  {
78      std::cout << prime << " is a prime number.";
79  }
```

nascardriver
May 18, 2019 at 11:12 pm · Reply

You're using recursion (A function calls itself (checkRemainder -> changeDiv -> checkRemainder -> ...)).
Every call needs to store some information about that call. The memory available for these information is limited, causing your program to abort once it's filled up. You'll later learn about loops, loops should be used instead of recursion.

> what is a trailing line feed
If you're running Windows, you might not be familiar with using a terminal (console). You probably run the application through your IDE or by double clicking it.
Linux/Mac tend to open a terminal, then run the application. The terminal stays open after the program has finished and can be re-used.

```
1  #include <iostream>
2
3  int main(void)
4  {
5    std::cout << "Hello world";
6
7    return 0;
8  }
```

Assuming '>' is the command prompt, running this program with './myProgram' will produce this output

```
1  > ./myProgram
2  Hello world>
```

The command prompt is now at the end of the last line printed, that's ugly.
Printing a line feed when your program is done solves this issue.

```
1  #include <iostream>
2
3  int main(void)
4  {
5    std::cout << "Hello world\n"; // added line feed
6
7    return 0;
8  }
```

```
1  > ./myProgram
2  Hello world
3  >
```

Now we can continue typing in the next line.

> what is double literal

```
1  // Those are integers
2  123
3  0
```

```
-43

// Those are doubles
123.0
0.0
-43.0

std::cout << (123 / -43) << '\n'; // Output: -2
std::cout << (123.0 / -43.0) << '\n'; // Output: -2.860465...
```

Jonathan
May 19, 2019 at 12:03 am · Reply

Thanks for the explanation! I get it now!

syobi
May 2, 2019 at 7:22 pm · Reply

```
bool isPrime(int x)
{

    if (x == 2)
    {
        return true;
    }

    if (x < 2 || x % 2 == 0)
    {
        return false;
    }

    for (int i = 3; i < x; i += 2)
    {
        if (x % i == 0)
        {
            return false;
        }
    }

    return true;
}
```

wrote isPrime(int x) that can also check other numbers too!