2.1 — Introduction to functions

BY ALEX ON MAY 31ST, 2007 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 27TH, 2020

In the last chapter, we defined a function as a collection of statements that execute sequentially. While that is certainly true, that definition doesn't provide much insight into why functions are useful. Let's update our definition: A **function** is a reusable sequence of statements designed to do a particular job.

You already know that every program must have a function named *main* (which is where the program starts execution when it is run). However, as programs start to get longer and longer, putting all the code inside the *main* function becomes increasingly hard to manage. Functions provide a way for us to split our programs into small, modular chunks that are easier to organize, test, and use. Most programs use many functions. The C++ standard library comes with plenty of already-written functions for you to use -- however, it's just as common to write your own. Functions that you write yourself are called **user-defined functions**.

Consider a case that might occur in real life: you're reading a book, when you remember you need to make a phone call. You put a bookmark in your book, make the phone call, and when you are done with the phone call, you return to the place you bookmarked and continue your book precisely where you left off.

C++ programs can work the same way. A program will be executing statements sequentially inside one function when it encounters a function call. A **function call** is an expression that tells the CPU to interrupt the current function and execute another function. The CPU "puts a bookmark" at the current point of execution, and then **calls** (executes) the function named in the function call. When the called function ends, the CPU returns back to the point it bookmarked, and resumes execution.

The function initiating the function call is called the **caller**, and the function being called is the **callee** or **called** function.

An example of a user-defined function

First, let's start with the most basic syntax to define a user defined function. For this lesson, all user-defined functions (except main) will take the following form:

```
return-type identifier() // identifier replaced with the name of your function
{
// Your code here
}
```

We'll talk more about the different parts of this syntax in the next few lessons. For now, *identifier* will simply be replaced with the name of your user-defined function. The curly braces and statements in between are called the **function body**.

Here is a sample program that shows how a new function is defined and called:

```
1  #include <iostream> // for std::cout
2
3  // Definition of user-defined function doPrint()
4  void doPrint() // doPrint() is the called function in this example
5  {
6      std::cout << "In doPrint()\n";
7  }
8
9  // Definition of function main()
10 int main()
11 {
```

```
    std::cout << "Starting main()\n";
    doPrint(); // Interrupt main() by making a function call to doPrint().  main() is the call
    std::cout << "Ending main()\n"; // this statement is executed after doPrint() ends

    return 0;
}
```

This program produces the following output:

```
Starting main()
In doPrint()
Ending main()
```

This program begins execution at the top of function *main*, and the first line to be executed prints `Starting main()`.

The second line in *main* is a function call to the function *doPrint*. We call function *doPrint* by appending a pair of parenthesis to the function name like such: do`Print()`. Note that if you forget the parenthesis, your program may not compile (and if it does, the function will not be called).

Warning

Don't forget to include parenthesis () after the function's name when making a function call.

Because a function call was made, execution of statements in *main* is suspended, and execution jumps to the top of called function *doPrint*. The first (and only) line in *doPrint* prints `In doPrint()`. When *doPrint* terminates, execution returns back to the caller (here: function *main*) and resumes from the point where it left off. Consequently, the next statement executed in *main* prints `Ending main()`.

Calling functions more than once

One useful thing about functions is that they can be called more than once. Here's a program that demonstrates this:

```
#include <iostream> // for std::cout

void doPrint()
{
    std::cout << "In doPrint()\n";
}

// Definition of function main()
int main()
{
    std::cout << "Starting main()\n";
    doPrint(); // doPrint() called for the first time
    doPrint(); // doPrint() called for the second time
    std::cout << "Ending main()\n";

    return 0;
}
```

This program produces the following output:

```
Starting main()
In doPrint()
```

```
In doPrint()
Ending main()
```

Since *doPrint* gets called twice by *main*, *doPrint* executes twice, and *In doPrint()* gets printed twice (once for each call).

Functions calling functions calling functions

You've already seen that function *main* can call another function (such as function *doPrint* in the example above). Any function can call any other function. In the following program, function *main* calls function *doA*, which calls function *doB*:

```
#include <iostream> // for std::cout

void doB()
{
    std::cout << "In doB()\n";
}


void doA()
{
    std::cout << "Starting doA()\n";

    doB();

    std::cout << "Ending doA()\n";
}

// Definition of function main()
int main()
{
    std::cout << "Starting main()\n";

    doA();

    std::cout << "Ending main()\n";

    return 0;
}
```

This program produces the following output:

```
Starting main()
Starting doA()
In doB()
Ending doA()
Ending main()
```

Nested functions are not supported

Unlike some other programming languages, in C++, functions cannot be defined inside other functions. The following program is not legal:

```
#include <iostream>

```

```
int main()
{
    int foo() // Illegal: this function is nested inside function main()
    {
        std::cout << "foo!\n";
        return 0;
    }

    foo(); // function call to foo()
    return 0;
}
```

The proper way to write the above program is:

```
#include <iostream>

int foo() // no longer inside of main()
{
    std::cout << "foo!\n";
    return 0;
}

int main()
{
    foo();
    return 0;
}
```

Quiz time

Question #1

In a function definition, what are the curly braces and statements in-between called?

Show Solution

Question #2

What does the following program print? Do not compile this program, just trace the code yourself.

```
#include <iostream> // for std::cout

void doB()
{
    std::cout << "In doB()\n";
}

void doA()
{
    std::cout << "In doA()\n";

    doB();
}

// Definition of function main()
int main()
{
    std::cout << "Starting main()\n";

    doA();
```

```
21      doB();
22
23      std::cout << "Ending main()\n";
24
25      return 0;
26  }
```

Show Solution

2.2 -- Function return values

Index

1.x -- Chapter 1 summary and quiz

C++ TUTORIAL | PRINT THIS POST

593 comments to 2.1 — Introduction to functions

« Older Comments 1 ... 6 7 8

Aryan Nair
February 7, 2020 at 5:02 am · Reply

The solution of question 2 should only have one "in doB" in the solution, as that function is only called once in the main body.

nascardriver
February 7, 2020 at 8:59 am · Reply

It's called again in `doA`.

Vitaliy Sh.
January 26, 2020 at 7:05 pm · Reply

[details="Hi Sires!"]
First: Can your, please, add details tag, like on ask.fedoraproject.org?

```
// "... function is/as a collection of statements ..."
// Which is correct English?
// In 1.1:
//      that executeS sequentially
// In 1.x:
//      that execute  sequentially
// here:
       "that execute  sequentially."


// Section diagram may be colored?
// Also: Can your, please:
// replace "void" with "return-type" here?
void identifier() // identifier replaced with the name of your function
{
// Your code here
}


// quotes, <em> ?
"prints Starting main()."
"prints In doPrint()"
"prints Ending main()."


// to remind that functions
// can be called not only from main():
// (here: function main)
"execution returns back to the caller (function main)"


// Can your please, add an ending '\n' to the Question #2 answer?
// People in comments have noted importance of last '\n' (to separate outputs of different
```

[/details]

prince
January 14, 2020 at 9:21 pm · Reply

What do you call the variable that comes before the "main ()".
I'm talking about the "int". And is that the only variable that we can put before the main ()? And is void is the only variable that we can use for a called function?

nascardriver
January 15, 2020 at 4:03 am · Reply

The `int` before `main` is the function's return type. You'll learn about this in the upcoming lessons.

Whims
January 10, 2020 at 8:39 pm · Reply

So I think I'm missing something, on the last question, how do you end up with a second doB()??? Maybe I'm just missing something super simple but I can't figure it out for the life of me.

flow
January 11, 2020 at 5:08 am · Reply

you just overlooked that doB() is actually called twice. doA() invokes doB() and the main function itself directly calls doB() afterwards.

Jim
December 18, 2019 at 2:24 pm · Reply

I setup Code Blocks as shown at the beginning of these lessons. On my D: drive. Now I have questions on how to use it.

How an I supposed to us these small program snippets?

First off it is setup for a project. So how do I set it up to run some of the snippets shown in these lessons alone?

I tried main menu File>New>File to run some of the snippets. But after I build and run them with C++, it just asks ne if I want to build them in a popup dialog box.

I used CB about 6 years ago and I don't recall using it in a project mode at all.

Note: I've been copying the snippet from these lessons and pasting them into the CB Editor/w a right click there, then us edit and paste there so I don't know it that the problem.

I give my files D:\<name> and C++.

How an I supposed to us these small program snippets.

Badreddine Boukheit
August 26, 2019 at 9:58 am · Reply

how can I download the whole series for offline exploration please x) !

David Cane
July 20, 2019 at 7:38 pm · Reply

I wonder if you can upload some vedios so that I can have a more graphic understanding of these lessons even though this may be not easy.But thanks anyway.

hunk head
June 22, 2019 at 8:08 am · Reply

umm excuse me but before you used "void x (x is whatever u name the function) ()" but then in the end
you used " int x as well" is there a specific difference betweenn the two?

nascardriver
June 22, 2019 at 8:37 am · Reply

`int x` is a variable, `void x()` is a function.

Samira Ferdi
June 10, 2019 at 5:15 pm · Reply

does function have address like a variable?

nascardriver
June 11, 2019 at 2:32 am · Reply

Yes, but they're stored in a part of memory that you can't (You're not supposed to) modify.

Samira Ferdi
June 11, 2019 at 5:08 pm · Reply

So, it means we can't take the address?

nascardriver
June 12, 2019 at 2:39 am · Reply

Yes you can, you just can't write to that address. Functions pointers are covered later.

Mohamed
May 7, 2019 at 8:31 pm · Reply

Thanks alot for this tutorial ,
that's too helpful and easy to understand.

Liam
April 9, 2019 at 9:24 am · Reply

I just wanted to say this tutorial is the shit, seriously. I've gone from VERY basic understanding of coding logic to feeling like I can competently learn C++ in about an hour. Thanks for taking the time to write all this!

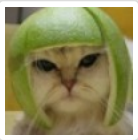
Pavlo
April 3, 2019 at 11:51 am · Reply

Hi! Thanks, for your work.
One remark: in the first example - line 14:

```
1 std::cout << "Ending main()\n" // this statement is executed after doPrint() ends
```

";" is missed.

Alex



April 3, 2019 at 1:20 pm · Reply

Fixed. Thanks!

Hemanth Kumar
April 2, 2019 at 8:08 am · Reply

Hi

can anybody please guide me, what is the mistake in the below program. am getting an error (am using Visual studio 2017)

```
#include <iostream>

void doPrint()
{
   std::cout << " In doPrint()\n";
}

//definition of function main()
int main();

{
   std::cout << "Starting main()\n";
   doPrint();
   std::cout << "Ending main()\n";

   return 0;
}
```

Error:

Severity Code Description Project File Line Suppression State
Error (active) E0169 expected a declaration Project1 C:\Users\Hemanth Kumar VS\source\repos\Project1\Project1\Project1.cpp 12

Severity Code Description Project File Line Suppression State
Error C2447 '{': missing function header (old-style formal list?) Project1 c:\users\hemanth kumar vs\source\repos\project1\project1\project1.cpp 12

nascardriver
April 3, 2019 at 2:38 am · Reply

Remove the semicolon behind main()

Hemanth Kumar
April 4, 2019 at 6:39 am · Reply

Thanks a lot…

Aakarsh
March 4, 2019 at 7:48 am · Reply

please guide me through this and help me solve it. I am unable to understand the error displayed, according to me there's no errors in this code but there's an error popping up all the time. Please Help. I am using https://ideone.com

Code Written

```
#include <iostream>
void Func2()
{
    std::cout<<"func2"std::endl;
}

int main()
{
    std::cout<<"starting int main()"<<std::endl;
    void Func2();
    void Func2();
    std::cout<<"ending int main()";
    return 0;
}
```

Error

```
Compilation error#stdin compilation error #stdout 0s 0KB
prog.cpp: In function ‘void Func2()’:
prog.cpp:4:13: error: unable to find string literal operator ‘operator""std’ with ‘const cha
  std::cout<<"func2"std::endl;
             ^~~~~~~~~~
```

nascardriver
March 4, 2019 at 7:52 am · Reply

You're missing << in line 4.

```
std::cout << "func2" << std::endl;
//                   ^^ Here
```

Aakarsh
March 4, 2019 at 9:26 pm · Reply

Thanks a lot

AARYN
May 17, 2019 at 2:27 am · Reply

Also in line 10 and 11 if you do not remove void, the called function will not respond.

Helliarc
February 20, 2019 at 6:53 pm · Reply

Maybe it has already been mentioned, but I tried and it didn't work for me… Are the functions built in sequence? I tried to call my second function with my first function (sequentially as declared), and function 1 failed to call function 2, but function 2 can call function 1… I imagine the program isn't going to load ALL of the functions, but regardless of it's position of definition, shouldn't calling a function cause the program to "search" for that function that it is trying to call?

```
1  void funcA()
2  {
3    std::cout << "funcA output \n";
4    funcB();  //attempt to call funcB from funcA, not working...
5  }
6
7  void funcB()
8  {
9    std::cout << "funcB output \n";
10 }
11
12 int main()
13 {
14   std::cout << "Main output \n";
15   funcA();
16
17   return 0;
18 }
```

nascardriver
February 21, 2019 at 8:18 am · Reply

Functions have to be declared before they can be used. This is covered later.

Helliarc
February 21, 2019 at 9:12 am · Reply

So if I call funcB in main, and THEN call funcA it will work??? I'm going to try that real quick before I move on today...

Helliarc
February 21, 2019 at 9:15 am · Reply

Nope, that didn't work... So it looks like the first function can't call subsequent functions in the list, but subsequent functions can call functions above it... I'm going to move on in the lessons and see if I answer my own question. Thanks for the reply!

Helliarc
February 21, 2019 at 3:11 pm · Reply

Figured it out!!! Forward declarations!

Oscar R.
February 17, 2019 at 1:20 am · Reply

Ah! Now this version of learncpp.com is what I should of had 1 month ago! Great improvement guys

GG
February 2, 2019 at 6:05 am · Reply

Alex, Relocating the DRY exercise from the chapter 1 Quiz (which was too tricky and so a bit demoralizing for newcomers) to an exercise in 2.2 section is a smart move...

Nice to see uniform initialization being moved into chapter one, instead of creeping in after people got bad habits in the old 2.4 if i remember correctly....
And so many other much needed improvements.....
I was actually going to write you about all these things, glad to see everything being updated....

in 1.10 compiler line error explanation for << being misused for cin do not match the actually example lines (due to the omission of #include <iostream>), this might confuse some.

Per our previous discussion: implementing top page navigation could be done with a simple small : < previous next > on the top right.

I would also advise to change your header banner/logo on desktop as it is dated, it gives the wrong impression and might scare some away.
Moreover, the title : learncpp.com and the your header: tutorials to help you master... are currently embedded as an image banner, these should be typed over your banner for SEO optimization, so search spiderscan read it and give you extra relevance.

(while I'm at it: the ad banners background on each side being grey is feeling old or dull and does not make the ads or the center section pop (=less ad clicks), a deeper grayish black would freshen it up but that's more of a taste thing)

Thank you for your time.

Alex
February 2, 2019 at 8:44 am · Reply

Thanks for all the feedback! The typos have been fixed. Still looking into the nav request. I'll look at the styling elements too. The site is overdue for a theme refresh but I'm not a designer so that's not my area of strength.

Tamir
February 5, 2019 at 4:29 pm · Reply

Hey, just want to say thanks for keeping up with all of this... this is great stuff and is really helping me out in my c++ endeavors. When I first saw some of the lessons I got worried bc some of the dates were old, but as I continued I saw updates being made so close to the current date. As a student, I just want to say I appreciate all of this and looking forward to completing the whole 'course' haha. Awesome stuff

Tamir

nascardriver
January 18, 2019 at 7:50 am · Reply

Hi Alex!
Quiz 1h says "What actually gets output depends on the compiler.".
I don't see how this is the case. To my understanding, it's well defined.
Function pointers can only be cast to bool, @std::basic_ostream::operator<< has an overload for bools, so that's used.

Kushal
January 18, 2019 at 5:42 am · Reply

"In the third function call of return5(), the function returns the value of 5 back to the caller. However, main() does nothing with the return value so nothing further happens (the return value

is ignored)."

What will happen with 5 if it is ignored. My question is in terms of memory. When a function returns a value then where is the existence of the value even though it is gonna be ignored/deleted ? Any temporary storage area in memory ?

Regards
Kushal

nascardriver
January 18, 2019 at 6:26 am · Reply

Hi Kushal!

This is implementation defined. Most likely, the called function (In this case @return5) stores the value in memory, but the memory is overridden later without ever being read from.

Kushal
January 18, 2019 at 7:33 am · Reply

Ok , thanks for the explanation :)

One more question. I just executed last example 1h in my case it always prints 1 when I run the program.

I replaced the rreturn5 with return10 to check but it gives error. So why does it works only for return5 ?

Not sure why.

nascardriver
January 18, 2019 at 7:44 am · Reply

> I just executed last example 1h in my case it always prints 1
It's supposed to do that.

> I replaced the rreturn5 with return10 to check but it gives error
There is nothing named "return10", so you get an error. If you add a function or variable with that name, you'll get the same output.

Kushal
January 18, 2019 at 8:58 am · Reply

ok , so if the function is not called properly , i.e without () then it will return 1 ?

nascardriver
January 18, 2019 at 9:00 am · Reply

fn() calls the function
fn is the function's memory address. @std::cout doesn't accept memory addresses of functions, so it converts it to a bool. Function addresses are not 0 and everything that is not 0 is true. True is represented by a 1.

Kushal
January 21, 2019 at 1:20 am · Reply

aaaahh, Thanks for the detailed explanation. Now I got everything clear.

« Older Comments 1 ... 6 7 8