2.1 — Introduction to functions

ALEX SEPTEMBER 13, 2022

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 executable 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 the next few lessons, all user-defined functions will take the following form:

```
return-type identifier() // This is the function header (tells the compiler about the existence of the function)
{
    // This is the function body (tells the compiler what the function does)
}
```

The first line is informally called the **function header**, and it tells the compiler about the existence of a function, what the function is called, and some other information that we'll cover in future lessons (like the return type and parameter types).

- In this lesson, we'll use a return-type of int (for function main()) or void (otherwise). We'll talk more about return types and return values in the next lesson (2.2 -- Function return values (value-returning functions)). For now, you can ignore these.
- Just like variables have names, so do user-defined functions. The identifier is the name of your user-defined function.
- The parentheses after the identifier tell the compiler that we're defining a function.

The curly braces and statements in-between are called the **function body**. This is where the statements that determine what your function does will go.

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

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

// Definition of user-defined function doPrint()
void doPrint() // doPrint() is the called function in this example
{
    std::cout << "In doPrint()\n";
}

// Definition of function main()
int main()
{
    std::cout << "Starting main()\n";
    doPrint(); // Interrupt main() by making a function call to doPrint().  main() is the caller.
    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 parentheses to the function name like such: `doPrint()`. Note that if you forget the parentheses, your program may not compile (and if it does, the function will not be called).

Warning

Don't forget to include parentheses () 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:

```
1  #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:

```cpp
#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:

```cpp
#include <iostream>

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

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

The proper way to write the above program is:

```cpp
#include <iostream>

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

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

As an aside...

"foo" is a meaningless word that is often used as a placeholder name for a function or variable when the name is unimportant to the demonstration of some concept. Such words are called metasyntactic variables (https://en.wikipedia.org/wiki/Metasyntactic_variable) (though in common language they're often called "placeholder names" since nobody can remember the term "metasyntactic variable"). Other common metasyntactic variables in C++ include "bar", "baz", and 3-letter words that end in "oo", such as "goo", "moo", and "boo").

For those interested in etymology (how words evolve), RFC 3092 (https://datatracker.ietf.org/doc/html/rfc3092) is an interesting read.

Quiz time

Question #1

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

Show Solution (javascript:void(0))

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();
    doB();

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

    return 0;
}
```

Show Solution (javascript:void(0))

Next lesson
2.2 Function return values (value-returning functions)

Back to table of contents

Previous lesson
1.x Chapter 1 summary and quiz

Leave a comment...

Name*

Email*

Find a mistake? Leave a comment!

Avatars from https://gravatar.com/ are connected to your provided email address.

Notify me about replies:

POST COMMENT

718 COMMENTS

Newest