1.4 — A first look at functions and return values

BY ALEX ON MAY 31ST, 2007 | LAST MODIFIED BY ALEX ON MAY 10TH, 2018

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). However, most programs use many functions.

Often, your program needs to interrupt what it is doing to temporarily do something else. You do this in real life all the time. For example, you might be 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 your book where you left off.

C++ programs 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 terminates, the CPU goes 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.

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

```cpp
//#include "stdafx.h" // Visual Studio users need to uncomment this line
#include <iostream> // for std::cout and std::endl

// Definition of function doPrint()
void doPrint() // doPrint() is the called function in this example
{
    std::cout << "In doPrint()" << std::endl;
}

// Definition of function main()
int main()
{
    std::cout << "Starting main()" << std::endl;
    doPrint(); // Interrupt main() by making a function call to doPrint().  main() is the caller.
    std::cout << "Ending main()" << std::endl;

    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(). At this point, execution of statements in main() is suspended, and the CPU jumps to doPrint(). The first (and only) line in doPrint prints `In doPrint()`. When doPrint() terminates, the caller (main()) resumes execution where it left off. Consequently, the next statement executed in main prints `Ending main()`.

Note that function calls are made by using the function name, plus an empty set of parenthesis (). We'll talk about what this set of parenthesis is in the next lesson. For now, just note that if you forget them, your program may not compile (and if it does, the function will not be called as expected).

Rule: Don't forget to include parenthesis () when making a function call.

Return values

When the main() function finishes executing, it returns an integer value (typically 0) back to the operating system (the caller) by using a return statement.

When you write your own functions, you get to decide whether a given function will return a value to the caller or not. This is done by setting the **return type** of the function in the function’s definition. The return type is the type declared before the function name. Note that the return type does not indicate what specific value will be returned. It only indicates what type of value will be returned.

Then, inside the called function, we use a return statement to indicate the specific value being returned to the caller. The actual value returned from a function is called the **return value**.

Let’s take a look at a simple function that returns an integer value, and a sample program that calls it:

```cpp
// int means the function returns an integer value to the caller
int return5()
{
    // this function returns an integer, so a return statement is needed
    return 5; // we're going to return integer value 5 back to the caller of this function
}

int main()
{
    std::cout << return5() << std::endl; // prints 5
    std::cout << return5() + 2 << std::endl; // prints 7

    return5(); // okay: the value 5 is returned, but is ignored since main() doesn't do anything with it

    return 0;
}
```

In the first function call of return5(), the function returns the value of 5 back to the caller, which passes that value to std::cout to be output.

In the second function call of return5(), the function returns the value of 5 back to the caller. The expression 5 + 2 is then evaluated to 7. The value of 7 is passed to cout to be output.

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).

Note: Return values will not be printed unless the caller sends them to std::cout. In the last case above, the return value is not sent to std::cout, so nothing is printed.

Return values of type void

Functions are not required to return a value. To tell the compiler that a function does not return a value, a return type of **void** is used. Let’s look at the doPrint() function from the program above:

```cpp
void doPrint() // void is the return type
{
    std::cout << "In doPrint()" << std::endl;
    // This function does not return a value so no return statement is needed
}
```

This function has a return type of void, indicating that it does not return a value to the caller. Because it does not return a value, no return statement is needed.

Here’s another example of a function that returns void, and a sample program that calls it:

```cpp
// void means the function does not return a value to the caller
void returnNothing()
{
    std::cout << "Hi" << std::endl;
    // This function does not return a value so no return statement is needed
}

int main()
{
    returnNothing(); // okay: function returnNothing() is called, no value is returned


    std::cout << returnNothing(); // error: this line will not compile.  You'll need to comment it out
 to continue.

    return 0;
}
```

In the first function call to returnNothing(), the function prints “Hi” and then returns nothing back to the caller. Control returns to main() and the program proceeds.

The second function call to returnNothing() won’t even compile. Function returnNothing() has a void return type, meaning it doesn’t return a value. However, function main() is trying to send this nothing value to std::cout to be printed. std::cout can’t handle “nothing” values, as it doesn’t know what to do with them (what value would it output?). Consequently, the compiler will flag this as an error. You’ll need to comment out this line of code in order to make your code compile.

A void return type (meaning nothing is returned) is typically used when we want to have a function that doesn’t return anything to the caller (because it doesn’t need to). In the above example, the returnNothing() function has useful behavior (it prints “Hi”) but it doesn’t need to return anything back to the caller (in this case, main()). Therefore, returnNothing() is given a void return type.

Returning to main

You now have the conceptual tools to understand how the main() function actually works. When the program is executed, the operating system makes a function call to main(). Execution then jumps to the top of main. The statements in main are executed sequentially. Finally, main returns an integer value (usually 0) back to the operating system. This is why main is defined as `int main()`.

Why return a value back to the operating system? This value is called a **status code**, and it tells the operating system (and any other programs that called yours) whether your program executed successfully or not. By consensus, a return value of 0 means success, and a positive return value means failure.

Note that the C++ standard explicitly specifies that main() must return an int. However, if you do not provide a return statement in main, the compiler will return 0 on your behalf. That said, it is best practice to explicitly return a value from main, both to show your intent, and for consistency with other functions (which will not let you omit the return value).

For now, you should also define your main() function at the bottom of your code file. We’ll talk about why shortly, in section **1.7 -- Forward Declarations**.

A few additional notes about return values

First, if a function has a non-void return type, it *must* return a value of that type (using a return statement). The only exception to this rule is for function main(), which will assume a return value of 0 if one is not explicitly provided.

Second, when a return statement is reached in a function, the function returns back to the caller immediately at that point. Any additional code in the function is ignored.

A function can only return a single value back to the caller. The function may, however, use any logic at its disposal to determine which specific value to return. It may return a single number (return 5). It may return the value of a variable or an expression (both of which evaluate to a single value). Or it may pick a single value from a set of possible values (which is still just a single value).

There are ways to work around only being able to return a single value back to the caller, which we will discuss when we get into the in-depth section on functions.

Finally, note that a function is free to define what its return value means. Some functions use return values as status codes, to indicate whether they succeeded or failed. Other functions return a calculated or selected value. Other functions return nothing. What the function returns and the meaning of that value is defined by the function’s author. Because of the wide variety of possibilities here, it’s a good idea to leave a comment on your functions indicating not just what they return, but also what the return value means.

Reusing functions

The same function can be called multiple times, which is useful if you need to do something more than once.

```
//#include "stdafx.h" // Visual Studio users need to uncomment this line
#include <iostream>

// getValueFromUser will read a value in from the user, and return it to the caller
int getValueFromUser()
```

```cpp
{
    std::cout << "Enter an integer: "; // ask user for an integer
    int a; // allocate a variable to hold the user input
    std::cin >> a; // get user input from console and store in variable a
    return a; // return this value to the function's caller (main)
}

int main()
{
    int x = getValueFromUser(); // first call to getValueFromUser
    int y = getValueFromUser(); // second call to getValueFromUser

    std::cout << x << " + " << y << " = " << x + y << std::endl;

    return 0;
}
```

This program produces the following output:

```
Enter an integer: 5
Enter an integer: 7
5 + 7 = 12
```

In this case, main() is interrupted 2 times, once for each call to getValueFromUser(). Note that in both cases, the value read into variable a is passed back to main() via the function's return value and then assigned to variable x or y!

Note that main() isn't the only function that can call other functions. Any function can call another function!

```cpp
//#include "stdafx.h" // Visual Studio users need to uncomment this line
#include <iostream>

void printA()
{
    std::cout << "A" << std::endl;
}

void printB()
{
    std::cout << "B" << std::endl;
}

// function printAB() calls both printA() and printB()
void printAB()
{
    printA();
    printB();
}

// Definition of main()
int main()
{
    std::cout << "Starting main()" << std::endl;
    printAB();
    std::cout << "Ending main()" << std::endl;
    return 0;
}
```

This program produces the following output:

```
Starting main()
A
B
Ending main()
```

Nested functions

Functions can not be defined inside other functions (called nesting) in C++. The following program is not legal:

```cpp
#include <iostream>

int main()
{
    void foo() // this function is nested inside main(), which is illegal.
    {
        std::cout << "foo!" << std::endl;
    }

    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!" << std::endl;
}

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

Quiz time

Inspect the following programs and state what they output, or whether they will not compile.

1a)

```cpp
#include <iostream>
int return7()
{
    return 7;
}

int return9()
{
    return 9;
}

int main()
{
    std::cout << return7() + return9() << std::endl;

    return 0;
}
```

Hide Solution

This program prints the number 16.

1b)

```cpp
#include <iostream>
int return7()
{
    return 7;

    int return9()
    {
        return 9;
    }
}

int main()
{
    std::cout << return7() + return9() << std::endl;

    return 0;
}
```

Hide Solution

This program will not compile. Nested functions are not allowed.

1c)

```
#include <iostream>
int return7()
{
    return 7;
}

int return9()
{
    return 9;
}

int main()
{
    return7();
    return9();

    return 0;
}
```

Hide Solution

This program compiles but does not produce any output. The return values from the functions are not used by main, and are thus discarded.

1d)

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

void printB()
{
    std::cout << "B" << std::endl;
}

int main()
{
    printA();
    printB();

    return 0;
}
```

Hide Solution

This program prints the letters A and B on separate lines.

1e)

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

int main()
{
    std::cout << printA() << std::endl;

    return 0;
}
```

Hide Solution

This program does not compile. Function printA() returns void, which main tries to send to std::cout. This will produce a compile error.

1f)

```
#include <iostream>
int getNumbers()
{
    return 5;
    return 7;
}

int main()
{
    std::cout << getNumbers() << std::endl;
    std::cout << getNumbers() << std::endl;

    return 0;
}
```

Hide Solution

This program prints 5 twice (on separate lines). Both times when function getNumbers() is called, the value 5 is returned. When the return 5 statement is executed, the function is exited immediately, so the return 7 statement never executes.

1g)

```
#include <iostream>
int return 5()
{
    return 5;
}

int main()
{
    std::cout << return 5() << std::endl;

    return 0;
}
```

Hide Solution

This program will not compile because the function has an invalid name.

1h) Extra credit:

```
#include <iostream>
int return5()
{
    return 5;
}

int main()
{
    std::cout << return5 << std::endl;

    return 0;
}
```

Hide Solution

This program will compile, but the function will not be called because the function call is missing parenthesis. What actually gets output depends on the compiler.

Share this:

510 comments to 1.4 — A first look at functions and return values

ztr36
July 5, 2018 at 12:50 am · Reply

I have developed a little further on the calculator in this lesson and I wanted to make it so the user could chose which operator to use so anyways here is my code

```
#include "stdafx.h"
#include <iostream>

int getValueFromUser()
{
    std::cout << "enter an integer: ";
    int a;
    std::cin >> a;
    return a;
}

char method()
{
    std::cout << "enter a operator: ";
    char b;
    std::cin >> b;
    return b;
}

int main()
{
    int x = getValueFromUser();
    char z = method();
```