# 2.2 — Function return values

ALEX NOVEMBER 6, 2020

Consider the following program:

```
#include <iostream>

int main()
{
 // get a value from the user
 std::cout << "Enter an integer: ";
 int num{};
 std::cin >> num;

 // print the value doubled
 std::cout << num << " doubled is: " << num * 2 <<
'\n';

 return 0;
}
```

This program is composed of two conceptual parts: First, we get a value from the user. Then we tell the user what double that value is.

Although this program is trivial enough that we don’t need to break it into multiple functions, what if we wanted to? Getting an integer value from the user is a well-defined job that we want our program to do, so it would make a good candidate for a function.

So let’s write a program to do this:

```
// This program doesn't work
#include <iostream>

void getValueFromUser()
{
  std::cout << "Enter an integer: ";
 int input{};
 std::cin >> input;
}

int main()
{
 getValueFromUser(); // Ask user for input

 int num{}; // How do we get the value from getValueFromUser() and use it to initialize this
variable?

 std::cout << num << " doubled is: " << num * 2 << '\n';

 return 0;
}
```

While this program is a good attempt at a solution, it doesn’t quite work.

When function *getValueFromUser* is called, the user is asked to enter an integer as expected. But the value they enter is lost when *getValueFromUser* terminates and control returns to *main*. Variable *num* never gets initialized with the value the user entered, and so the program always prints the answer 0.

What we're missing is some way for *getValueFromUser* to return the value the user entered back to *main* so that *main* can make use of that data.

## Return values

When you write a user-defined function, you get to determine whether your function will return a value back to the caller or not. To return a value back to the caller, two things are needed.

First, your function has to indicate what type of value will be returned. This is done by setting the function's **return type**, which is the type that is defined before the function's name. In the example above, function *getValueFromUser* has a return type of *void*, and function *main* has a return type of int. Note that this doesn't determine what specific value is returned -- only the type of the value.

Second, inside the function that will return a value, we use a **return statement** to indicate the specific value being returned to the caller. The specific value returned from a function is called the **return value**. When the return statement is executed, the return value is copied from the function back to the caller. This process is called **return by value**.

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

```cpp
#include <iostream>

// int is the return type
// A return type of int means the function will return some integer value to the caller (the specific value is not specified here)
int returnFive()
{
    // the return statement indicates the specific value that will be returned
    return 5; // return the specific value 5 back to the caller
}

int main()
{
    std::cout << returnFive() << '\n'; // prints 5
    std::cout << returnFive() + 2 << '\n'; // prints 7

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

    return 0;
}
```

When run, this program prints:

```
5
7
```

Execution starts at the top of main. In the first statement, the function call to *returnFive* is evaluated, which results in function *returnFive* being called. Function *returnFive* returns the specific value of 5 back to the caller, which is then printed to the console via std::cout.

In the second function call, the function call to *returnFive* is evaluated, which results in function *returnFive* being called again. Function *returnFive* returns the value of 5 back to the caller. The expression 5 + 2 is evaluated to produce the result 7, which is then printed to the console via std::cout.

In the third statement, function *returnFive* is called again, resulting in the value 5 being returned back to the caller. However, function *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 the console via std::cout. In the last case above, the return value is not sent to std::cout, so nothing is printed.

## Fixing our challenge program

With this in mind, we can fix the program we presented at the top of the lesson:

```cpp
#include <iostream>

int getValueFromUser() // this function now returns an integer value
{
  std::cout << "Enter an integer: ";
 int input{};
 std::cin >> input;

 return input; // return the value the user entered back to the caller
}

int main()
{
 int num { getValueFromUser() }; // initialize num with the return value of
getValueFromUser()

 std::cout << num << " doubled is: " << num * 2 << '\n';

 return 0;
}
```

When this program executes, the first statement in *main* will create an int variable named *num*. When the program goes to initialize *num*, it will see that there is a function call to *getValueFromUser*, so it will go execute that function. Function *getValueFromUser*, asks the user to enter a value, and then it returns that value back to the caller (*main*). This return value is used as the initialization value for variable *num*.

Compile this program yourself and run it a few times to prove to yourself that it works.

## Void return values

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 previous lesson:

```cpp
void doPrint() // void is the return type
{
    std::cout << "In doPrint()" << '\n';
    // 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 (trying to return a specific value from a function with a void return type will result in a compilation error).

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

```
#include <iostream>

// void means the function does not return a value to the caller
void returnNothing()
{
    std::cout << "Hi" << '\n';
    // 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 function *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 function *returnNothing* won’t even compile. Function *returnNothing* has a void return type, meaning it doesn’t return a value. However, this statement is trying to send the return value of *returnNothing* to std::cout to be printed. std::cout doesn’t know how to handle this (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 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), and your program terminates. The return value from `main` is sometimes called a **status code** (also sometimes called an **exit code**, or rarely a**return code**), as it is used to indicate whether the program ran successfully or not.

By definition, a status code of 0 means the program executed successfully.

### Best practice

Your *main* function should return 0 if the program ran normally.

A non-zero status code is often used to indicate failure (and while this works fine on most operating systems, strictly speaking, it's not guaranteed to be portable).

### For advanced readers

The C++ standard only defines the meaning of 3 status codes: 0, EXIT_SUCCESS, and EXIT_FAILURE. 0 and EXIT_SUCCESS both mean the program executed successfully. EXIT_FAILURE means the program did not execute successfully.

EXIT_SUCCESS and EXIT_FAILURE are defined in the <cstdlib> header:

```
#include <cstdlib> // for EXIT_SUCCESS and
EXIT_FAILURE

int main()
{
    return EXIT_SUCCESS;
}
```

If you want to maximize portability, you should only use 0 or EXIT_SUCCESS to indicate a successful termination, or EXIT_FAILURE to indicate an unsuccessful termination.

C++ disallows calling the *main* function explicitly.

For now, you should also define your*main* function at the bottom of your code file, below other functions.

## 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). Failure to do so will result in undefined behavior. The only exception to this rule is for function main(), which will assume a return value of 0 if one is not explicitly provided. 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).

### Best practice

Always explicitly provide a return value for any function that has a non-void return type.

**Warning**

Failure to return a value from a function with a non-void return type (other than main) will result in undefined behavior.

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

Third, a function can only return a single value back to the caller each time it is called. However, the value doesn't need to be a literal, it can be the result of any valid expression, including a variable or even a call to another function that returns a value. In the *getValueFromUser()* example above, we returned a variable holding the number the user typed.

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 document your function with a comment indicating what the return values mean.

For example:

```
// Function asks user to enter a value
// Return value is the integer entered by the user from the keyboard
int getValueFromUser()
{
  std::cout << "Enter an integer: ";
 int input{};
 std::cin >> input;

 return input; // return the value the user entered back to the caller
}
```

## Reusing functions

Now we can illustrate a good case for function reuse. Consider the following program:

```cpp
#include <iostream>

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

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

 std::cout << x << " + " << y << " = " << x + y << '\n';

 return 0;
}
```

While this program works, it’s a little redundant. In fact, this program violates one of the central tenets of good programming: “Don’t Repeat Yourself” (often abbreviated “DRY”).

Why is repeated code bad? If we wanted to change the text “Enter an integer:” to something else, we’d have to update it in two locations. And what if we wanted to initialize 10 variables instead of 2? That would be a lot of redundant code (making our programs longer and harder to understand), and a lot of room for typos to creep in.

Let’s update this program to use our *getValueFromUser* function that we developed above:

```cpp
#include <iostream>

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

 return input;
}

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

    std::cout << x << " + " << y << " = " << x + y << '\n';

    return 0;
}
```

This program produces the following output:

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

In this program, we call *getValueFromUser* twice, once to initialize variable *x*, and once to initialize variable *y*. That saves us from duplicating the code to get user input, and reduces the odds of making a mistake. Once we know *getValueFromUser* works for one variable, it will work for as many of them as we need.

This is the essence of modular programming: the ability to write a function, test it, ensure that it works, and then know that we can reuse it as many times as we want and it will continue to work (so long as we don't modify the function -- at which point we'll have to retest it).

**Best practice**

Follow the DRY best practice: "don't repeat yourself". If you need to do something more than once, consider how to modify your code to remove as much redundancy as possible. Variables can be used to store the results of calculations that need to be used more than once (so we don't have to repeat the calculation). Functions can be used to define a sequence of statements we want to execute more than once. And loops (which we'll cover in a later chapter) can be used to execute a statement more than once.

## Conclusion

Return values provide a way for functions to return a single value back to the function's caller.

Functions provide a way to minimize redundancy in our programs.

## Quiz time

**Question #1**

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() << '\n';

    return 0;
}
```

Show Solution

1b)

```cpp
#include <iostream>

int return7()
{
    return 7;

    int return9()
    {
        return 9;
    }
}

int main()
{
    std::cout << return7() + return9() << '\n';

    return 0;
}
```

Show Solution

1c)

```cpp
#include <iostream>

int return7()
{
    return 7;
}

int return9()
{
    return 9;
}

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

    return 0;
}
```

Show Solution

1d)

```cpp
#include <iostream>

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

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

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

    return 0;
}
```

Show Solution

1e)

```cpp
#include <iostream>

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

int main()
{
    std::cout << printA() << '\n';

    return 0;
}
```

Show Solution

1f)

```cpp
#include <iostream>

int getNumbers()
{
    return 5;
    return 7;
}

int main()
{
    std::cout << getNumbers() << '\n';
    std::cout << getNumbers() << '\n';

    return 0;
}
```

Show Solution

1g)

```cpp
#include <iostream>

int return 5()
{
    return 5;
}

int main()
{
    std::cout << return 5() << '\n';

    return 0;
}
```

Show Solution

1h) Extra credit:

```cpp
#include <iostream>

int returnFive()
{
    return 5;
}

int main()
{
    std::cout << returnFive << '\n';

    return 0;
}
```

Show Solution

**Question #2**

What does “DRY” stand for, and why is it a useful practice to follow?

Show Solution

**Next lesson**
2.3 Introduction to function parameters and arguments

**Back to table of contents**

**Previous lesson**
2.1 Introduction to functions

Leave a comment... Put C++ code between triple-backticks (markdown style):```Your C++ co

Name*

Email*

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

Notify me about replies:

POST COMMENT

DP N N FOUT

Newest

X

X