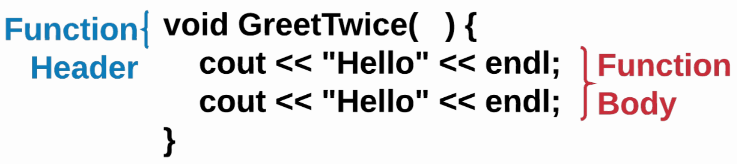
Learning Objectives: Function Basics

- **Demonstrate how to define a function**
- **Differentiate between the function header and the function body**
- **Identify the importance of the order of function definitions**
- **Identify the purpose of C++ code documentation**

Function Definition

Function Syntax

You have seen and used built-in functions like the length function (`my_string.length()`). This unit deals with user-defined functions. Functions are composed of two parts, the header and the body.

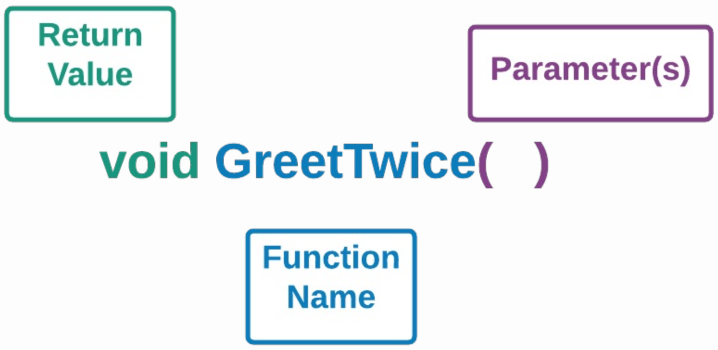


```
void GreetTwice(   ) {
    cout << "Hello" << endl;
    cout << "Hello" << endl;
}
```

.guides/img/FunctionFull

The function header may contain several keywords to determine the function type. Next is the name of the function. “Ordinarily, functions should start with a capital letter and have a capital letter for each new word.” (https://google.github.io/styleguide/cppguide.html#Function_Names). Some examples include: `GreetTwice`, `AddThree`, `FindArea`, etc. Parentheses are required but the parameter(s) within them are not. Any command(s) associated with the function should be indented as a best practice and enclosed within curly braces `{}`. This command(s) comprises the body of the function.

Function Header

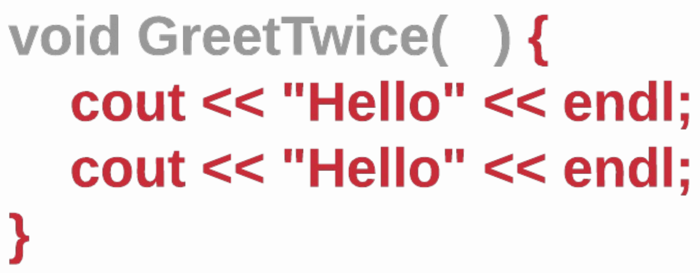


```
void GreetTwice(   )
```

.guides/img/FunctionHeader

The function header usually contains a few key labels:
* `void` - Determines whether there is a return value or not for the function. If the function has no return value, use **void**. If the function returns an integer, use **int**, etc.
* `GreetTwice` - This is an example of a function name. See above for naming conventions.
* `()` - Parentheses are required for all functions. Any parameters that the function takes in will go into the parentheses but they are optional.

Function Body



```
void GreetTwice(   ) {
    cout << "Hello" << endl;
    cout << "Hello" << endl;
}
```

.guides/img/FunctionBody

The function body is the list of actions the function performs. All of the code for the function body should be enclosed within curly braces `{}` and indented as a best practice to show association. This convention is similar to how conditionals and loops are written.

Function Calls

Calling a Function

Copy the entire code below into the text editor to your left. Then click the `TRY IT` button to see what happens.

```
#include <iostream>
using namespace std;

void GreetTwice() {
  cout << "Hello" << endl;
  cout << "Hello" << endl;
}
```

You'll notice that an error is produced. This happens because when running C++ programs, a `main` function is required. Let's go ahead and add `int main()` to the code like so.

```
#include <iostream>
using namespace std;

void GreetTwice() {
  cout << "Hello" << endl;
  cout << "Hello" << endl;
}

int main() {

}
```

Nothing is outputted when the program is executed. This happens because creating a function alone does not cause C++ to run it. You have to explicitly call the function if you want it to run. Functions are usually called within the `main` function. To call a function, simply start with its name, provide any parameters if needed, and end with a semicolon.

```
#include <iostream>
using namespace std;

void GreetTwice() {
  cout << "Hello" << endl;
  cout << "Hello" << endl;
}

int main() {
  GreetTwice();
}
```

challenge

What happens if you:

- Call `GreetTwice` again by adding another `GreetTwice();` to the `main()` function?
- Modify `GreetTwice` by adding the line `cout << "Goodbye" << endl;` underneath the second `cout << "Hello" << endl;`?
- Add the command `return 0` as the last line in the `main()` function?

important

IMPORTANT

You may have noticed that adding `return 0` to `main()` in the code above doesn't really cause any changes to the program. In older versions of C++, the command `return 0` was required because the `main()` function has a return type of `int`. This means that the function expects some integer value to be returned at the end of the function. `return 0` is another way for the program to denote that the function has successfully completed.

Newer versions of C++ can run `int main()` without `return 0`, however, it is still a best practice to include it. Additionally, you **should not** change the return type of `main()` to anything other than `int`. This is a standard in C++ and should remain so.

Order of Function Definitions

The order of function definitions is important in C++. If the code is changed to the following, what do you think will be outputted?

```
#include <iostream>
using namespace std;

void GreetTwice() {
  cout << "Goodbye" << endl;
  cout << "Hello" << endl;
  cout << "Hello" << endl;
}

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

Like how a regular C++ program runs, the function is executed line by line from top to bottom. Thus, the order of statements within the function will determine what actions are performed first, second, etc.

Order of Different Functions

The order of the functions themselves also matter. What happens if you swap the position of `main()` with `GreetTwice()` and vice versa?

```
#include <iostream>
using namespace std;

int main() {
  GreetTwice();
  GreetTwice();
  return 0;
}

void GreetTwice() {
  cout << "Goodbye" << endl;
  cout << "Hello" << endl;
  cout << "Hello" << endl;
}
```

You will encounter an error such as `GreetTwice was not declared in this scope`. Like how function definitions are read line by line from top to bottom, functions are also read the same way. When the program gets to the first `GreetTwice();` within `main()`, it doesn't know what to do because the function has not been declared yet. Therefore, whenever you call a function, make sure that you have already declared and defined it. Think of a function as a vocabulary word, you would not use a vocabulary word that you do not know the definition of.

Documentation

C++ Code Documentation

Including C++ code documentation prior to function definitions is standard. Doing so enables users to gain clarity on the purpose of the function, any parameters that are used, and what the function returns. Users can also see who wrote the function as well as what version the function is on. There are several ways to document C++ code, however the style that we will be using for this module models that of Java's (also known as Javadoc). Here is an example, also present within the text editor to your left:

```cpp
/**
 * This is an example of C++ documentation
 *
 * @author  FirstName LastName
 * @version 1.0
 */
#include <iostream>
using namespace std;

/**
 * This function greets the user twice
 *
 * @param   specify parameters if any
 * @return  specify return value if any
 */
void GreetTwice() {
  cout << "Hello" << endl;
  cout << "Hello" << endl;
}

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

The C++ code documentation does not affect the output of the code. However, it provides more clarity to how the function is used. Generally speaking, the documentation for authors and program version goes at the

start of the program while the documentation for parameters and return value goes directly **before** the declared function. Note that documentation for the `main()` function is not required.

Doxygen Tool

There is an online tool called **Doxygen** that can help generate C++ code documentation. For more information, you may visit the Doxygen website at this link: Doxygen. However, for the purposes of this module, we will only focus on self-created documentation. In particular, we will be using mostly `@param` for parameters and `@return` for return values to document our functions.