# 01-Structure of a Program

Before delving into C++, it's important to understand how C++ programs are structured. A C++ program (and most programs in other languages) are comprised of three primary things:

- Imports (i.e., `#include`)
- Statements
  - Comments
- Functions (an Overview)
  - `main()` Function

By the end of this chapter, we'll be able to make sense of this basic "Hello world" program:

```
#include <iostream>

int main() {
        std::cout << "Hello, world!\n";
        return 0;
}
```

Copy

## Imports (`#include`)

In order for C++ programs to be scalable -- and frankly writable at all -- it generally helps if you're able to write code into separate file modules and use them in multiple other files. For example, the *C++ standard library* includes objects like `std::cout`, `std::cin` and various functions to read and write from the console.

In order to use external code in your own program -- such as for accessing objects like `std::cout` in our sample program -- we must include the *header file* from which code modules are defined. The `iostream` header gives us access to the C++ standard library's `std::cout` object for writing data to the console. Including `iostream` is done via:

```
#include <iostream>
```

For advanced readers:

> The `#include` tag is called a *preprocessor directive*, and is used to copy the source of a specified file (usually a header file) into the current file. The preprocessor is

> covered in a later section.

There are various other header files provided by the C++ standard library, such as:

- `string` - for `std::string` and string operations
- `array` - for C++ arrays
- `vector` - for resizeable arrays
- `algorithm` - for various utility functions
- `cstdio` / `cstdint` / `cstdlib` /etc. - for C++ bindings over the C standard library
- Many more that you wont learn about in class.

> These may or may not be covered in later sections.

# Statements

In C++, a *statement* -- put simply -- is a unit of code that is intended to perform an action. A statement can include printing some text to the console, assigning a value to a variable, declaring a function, etc. The following snippet from our sample program is a statement:

```
        std::cout << "Hello, world!\n";
```

This statement specifically writes `"Hello, world!\n"` to `std::cout`.

Other examples of statements include:

```
int x; // variable declaration
x = 5; // variable definition
std::string name = "Emmett Tomai"; // inline variable definition
print_name(name); // function invocation
```

You might notice that all of these statements have one thing in common (besides the forward slashes): they all end with a semicolon character ( `;` ). All statements must end with semicolons.

## Comments

The text after the double-forward-slash ( `//` ) on each line of the above snippet is called a *comment*, which is a segment of text that is ignored by the compiler. Comments that start with a `//` continue to the end of the current line. You may also write either in-line or multi-line comments via `/* ... */`:

```
int x = 5; /* This is a multiline comment.
                          notice that it continues until
                          this symbol -> */
int y /* you can write them in-line as well */ = x + 1;
```

# Functions (an Overview)

Another important part of a program that we want to be useful (scalable and writable) is the ability to label and repeat blocks of code. *Functions* are blocks of code that can be executed from anywhere in a program.

Functions have two categories: declarations and definitions. A *function declaration* tells the compiler that certain function exists, so that when it is *invoked* by some statement in the program, the compiler is aware of its existence.

While a function declaration tells the compiler about the existence of a certain function symbol, functions need to be *defined* with a series of statements that describe the execution of the function. The following from our sample program is considered a function definition:

```
int main() {
        std::cout << "Hello, world!\n";
        return 0;
}
```

**In a later section, we will delve into the significance of functions and how they're best utilized by C++ programs. For now, it's imperative to understand what a function is to know what `int main() { ... }` means.**

## `main()` Function

`int main();` is a function which every C++ program must have. The beginning of `main()` marks the beginning of execution in a C++ program.

> It should be noted that while `main` *is* a function, it should *never* be called within your program. `main` is the only function to which this rule applies to, making it an exception to the semantics of function invocation.

You may notice that at the end of our `main()` function is a statement ***`return 0;`***. `return` is a keyword in C++ that **terminates the execution of a function and sends the defined value back to the *caller*** (the segment of code that invoked a function). In the case of `main()`, returning marks the end of program execution, and the *exit code* of the program is set to whatever value is returned by `main()`.

Lastly, `main()` **can be defined only once in your program. This is the case for all function definitions, which we will cover in** `06-Functions`

# Exercise - For advanced (or curious) readers

As an exercise, return any integer number you want from `main()` and view the result of your program via the command line.

> On Linux and MacOS, you can view the exit code of the last process via `echo $?`. For Windows users, you may view the exit code of the last process executed within the command prompt via `echo %ERRORLEVEL%`.

Furthermore, feel free to play with the program and see what you can do before moving on. It is encouraged that you do this after each section to see what you can come up with.

# What's Next?

In the next section, we will learn how to write programs that users can run and use.

---

Courtesy of *DeCentralize the Web* (2024)