C++ Primer Notes

Kat

March 10, 2020

Chapter 1

Ch. 01: Getting Started

1.1 Introduction

Every C++ program has one or more functions, with one of these functions being `main()`. A function is defined with four parts:

R.T.	Return type
F.N.	Function type
P.L.	Parameter list, maybe empty
F.B.	Function body, inside braces

For example, a basic `main()` function would look like this:

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

`int` is the return type, with the function `main` requiring an `int` R.T. A semicolon (`;`) closes a statement inside a function. For `main()`, the function returns a status indicator. Thus `return` is required, with `0` indicating a success.

Every data element (called objects in C++) must have a type. The type lets the compiler know what operations are possible on the object. For instance, say we have a variable `v` and the type of `v` is `T`. It would be described as "`v` has type `T`" or "`v` is a `T`". Types are integral to C++ and must always be given for any object.

1.2 Compiling

C++ is a compiled language, meaning that a compiler is required to take the human friendly language to something a computer can understand. This is in contrast to a language like Python in which the language you write in is the language that is run.

1.2.1 The GNU Compiler Collection: GCC

Since I am using Linux, the primary C++ compiler of use is GCC. While this has other compilers for different languages, we are concerned with `G++`. For simple programs, the primary usage is as follows:

```
$ g++ -o output input.cpp
```

1.3 Input/Output

C++ doesn't natively handle input and output operations but relies on a built in library called `iostream`. C++ gets input/output data via a stream, a sequence of characters read or written to an IO device that is generated or consumed sequentially. In the `iostream` library there are two types of streams: `istream` and `ostream`. There are a handful of IO objects in this library that we can classify:

Function	Use	Note
`cin`	Standard input	Type `istream`
`cout`	Standard output	Type `ostream`
`cerr`	Standard error	For general errors
`clog`	Standard log	For general info on the program

1.4 Namespaces

C++ has many functions, and some share names between libraries. The compiler and author have to know what object one is referring to. To do this, we prepend a namespace to the object in question and link them with a scope operator:

```
std::cout
```

`std` is the standard C++ namespace and most objects in the standard libraries use this namespace. `::` is the scope operator and it lets us describe a namespace within a scope.

1.4.1 Headers

A header links to a library and we use them in C++ programs via the `#include` director. This is used outside of the function and tells the compiler to include the library while compiling. It is used like so:

```
#include <iostream>
```

1.5 Comments

Comments are integral to any programming language. They improve readability and help the author and people reading the code to better understand the code at hand. In C++ there are two kinds of comments: single-line and paired.

1.5.1 Single-line Comments

Single-line comments are made with two forwardslashes, `//`. Everything past this is not read by the compiler up until a newline is made. For example:

```
std::cout;; // this comment keeps the code in view of the compiler
// std::cout;
// in the line above, the code is commented out
```

1.5.2 Paired Comments

A paired comment lets one create large blocks of comments, particularily on multiple lines, without having to use single-line comments for each line. A paired comment is started with `/*` and ends on the *first* instance of `*/`. This last part is important and means we can't nest paired comments. If we wanted to comment out a section of code that contains a set of paired comments, we would be unable to. For instance:

```
/* we start our comment here
stuff /* paired */
we end our paired here */
```

In this example the paired comment ends in the second line at the first `*/`. This leaves the second `*/` without an initial `/*`. This block would thus be invalid. And so in order to comment out paired comments, one should use single-line comments for every line involved.

1.6 The *for* Statement

In C++, `while` loops are very common. The most common of these are `while` loops that increment a value until it reaches a condition set by the author:

```
while ( i < 10 ) {
        do stuff;
        ++i
}
```

Since this `while` loop is so prominent, C++ introduced a new function to replicate it simpler: the `for` loop. A `for` loop contains three parts in its header: a init statement, a condition, and an expression.

Part	Example	Description
Init statement	`int val = 1;`	Defines a variable for the loop
Condition	`val <= 10;`	Describes when to end the loop
Expression	`++val`	What to do after each loop

And all together this would become:

```
for ( int val = 1; val <= 10; ++val) {
        stuff;
        maybe more stuff;
}
```

It is important to note that only the first two parts are ended by a semicolon, the expression is not ended by a semicolon.

1.7 Data Structures

In C++ we often want to be able to define our own objects, types, and functions. This is, in fact, what makes C++ so powerful. We can arbitrarily add in our own classes that behave like standard classes. A data class defines a type along with a collection of operations related to that type. In order to include these we must have a file (typically '*.h') and include it into our program. We can do that with `#include "Our_class.h"`.