C++ Programming: The C++ programming language

Introducing C++

C++ (pronounced "see plus plus") is a *general-purpose*, *multi-paradigm*, *statically typed*, *free-form* ***programming language***, supporting procedural; object-oriented; generic; and (more recently) functional programming paradigms, and is well-known for facilitating low-cost abstractions in code. If any of the preceding concepts are unfamiliar to you, do not worry, they will be introduced in subsequent sections.

During the 1990s **C++** grew to become one of the most popular computer programming languages, and it is still the third most popular language, according to the Tiobe rankings (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html) as of November 2017. **C++** was first designed with a focus on systems programming, but its features also make it an attractive language for creating end-user applications, especially those with resource constraints, or that require very high performance. **C++** is extensively used in game development, web clients/server side, back office of financial applications and robotics.

History and standardization

Bjarne Stroustrup, a Computer Scientist from Bell Labs, was the designer and original implementer of **C++** (originally named "C with Classes") during the 1980s, as an enhancement to the C programming language. C, which had also been created at Bell Labs for the purpose of implementing the Unix operating system by Dennis Ritchie, gave users great control over hardware at a higher conceptual level than assembly language (ASM), but still with limited expressivity. Stroustrup decided to combine features for program organization from the object-oriented Simula language with C's efficient use of hardware resources. Enhancements started with the addition of object-oriented concepts like classes and virtual functions, followed by, among many features, namespaces, operator overloading, templates, and exception handling. These and other features are covered in detail in this book. Several features of C++ were later adopted by C, including the `const` keyword for creating immutable values in a program, `inline` functions, declarations in `for loops` , and **C++**-style comments (using the `//` symbol).

The **C++ programming language** is a standard recognized by the ANSI (The American National Standards Institute), BSI (The British Standards Institute), DIN (The German national standards organization), and several other national standards bodies, and was ratified in 1998 by the ISO (The International Standards Organization) as *ISO/IEC 14882:1998*. The standard consists of two parts: the Core Language and the Standard Library; the latter includes the Standard Template Library and the Standard C Library (ANSI C 89).

The 2003 version, *ISO/IEC 14882:2003*, redefined the standard language as a single item. The STL ("Standard Template Library") that pre-dated the standardization of **C++** (and was originally implemented in Ada) became an integral part of the standard, and a requirement for a compliant implementation of the same.

From 2004, the standards committee (which includes Bjarne Stroustrup) worked out the details of a new revision of the standard, with **C++11** (also called **C++0x**) approved on 12 August 2011. **C++11** made the language more efficient, easier to use, and added more functionality to the Standard Library. The specification for **C++14** was released on 15 December 2014, with smaller changes compared to **C++11**, and compiler support for this standard has followed quickly. Several tables (http://en.cppreference.com/w/cpp/compiler_support) of compiler support for so-called **modern C++** features are available.

Many other **C++** libraries exist which are not part of the Standard, a popular example being Boost. Also, non-Standard libraries written in C can generally be used by **C++** programs.

C++ source code example

```
// 'Hello World!' program

#include <iostream>

int main()
{
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

Traditionally, the first program people write in a new language is called "Hello World", because all it does is simply display the words **Hello World**, while revealing basic information about the language in the process. Hello World Explained (in the Examples Appendix) offers a detailed explanation of this code, in which can be seen several elements of **C++** mentioned here, including C-like syntax and use of the Standard Library.

Overview

Before you begin your journey to understand how to write programs using **C++**, it is important to understand a few key concepts that you may encounter. These concepts are not unique to **C++**, but are helpful to understanding computer programming in general. Readers who have experience in another programming language may wish to skim through this section, or skip it entirely.

There are many different kinds of programs in use today. From the operating system you use that makes sure everything works as it should, to the video games and music applications you use for entertainment, programs can fulfill many different purposes. What all **programs** (also called **software** or **applications**) have in common is that they all are made up of a sequence of instructions written, in some form or another, in a programming language. These instructions tell a computer what to do, and generally how to do it. Programs can contain anything from instructions to solve math problems, to how to behave when a video game character is shot in a game. The computer will follow the instructions of a program one instruction at a time from start to finish.

Another thing true of all computer programs (or most programs, rather) is that they solve problems and perform tasks. Say hello to the world. Paint a button on the screen. Calculate 26*78. Drive the car. Fortunately or not, computers must be taught how to perform these tasks. In other words, they must be programmed.

Why learn C++ ?

Why not? This is the most clarifying approach to the decision to learn anything. Although learning is always good, selecting what you learn is more important as it is how you will prioritize tasks. Another side of this problem is that you will be investing some time in getting a new skill set. You must decide how this will benefit you. Check your objectives and compare similar projects or see what the programming market is in need of. In any case, the more programming languages you know, the better.

C++ is not the ideal first language. However, if you are willing to dedicate a more than passing interest in **C++**, then you can even learn it as your first language. Make sure to dedicate some time understanding the different paradigms and why C++ is a multi-paradigm, or hybrid, language.

If you are approaching the learning process only to add another notch under your belt, that is, willing only to dedicate enough effort to understand its major quirks and learn something about its dark corners, then you would be best served in learning two other languages first. This will clarify what makes **C++** special in its approach to programming. You should select one imperative and one object-oriented language. C will probably be the best choice for the former, as it has a good market value and a direct relation to **C++**, although a good substitute would be ASM. For the latter, Java is a good choice, mostly because it shares much of its syntax with C++ but it does not support imperative programming. Read the language comparison section for better understanding the relations.

Although learning C is not a requirement for understanding **C++**, you must know how to use an imperative language. **C++** will not make it easy for you to understand some of these deeper concepts, since in it you, the programmer, are given the greater range of freedom. There are many ways of doing things in C++. Understanding which options to choose will become the cornerstone of mastering the language.

You should not learn **C++** if you are solely interested in learning Object-oriented Programming. C++ offers some support for objects, but is still not truly Object-oriented, and consequently the nomenclature used and the approaches taken to solve problems will make it more difficult to learn and master those concepts. If you are truly interested in Object-oriented programming, you should learn Smalltalk.

As with all languages, **C++** has a specific scope of application where it can truly shine. C++ is harder to learn than C and Java but more powerful than both. **C++** enables you to abstract from the little things you have to deal with in C or other lower level languages but will grant you more control and responsibility than Java. As it will not provide the default features you can obtain in similar higher level languages, you will have to search and examine several external implementations of those features and freely select those that best serve your purposes (or implement your own solution).
