Beginning C++ Programming - From Beginner to Beyond

David Corzo

2020 May 18

Contents

Chapter 1

Introduction

1.1 Why learn C++?

- Popular:
  - Lots of code is still written in C++.
  - Programming language popularity indexes ranks C++ high.
  - Active community, GitHub, Stack overflow.
- Relevant:
  - Windows, Linux, MaxOSX, Photoshop, Illustrator, MySQL, MongoDB.
  - Amazon, Apple, Microsoft, PayPal, Google, Facebook, MySQL, Oracle, HP, IBM, more...
  - VR, Unreal Engine, Machine learning, networking & telecom, more...
- Powerful:
  - Super-fast, scalable, portable.
  - Supports both procedural and object-oriented programming.
- Good career opportunities:
  - C++ skills always in demand.
  - C++ = Salary++.

1.2 Modern C++ and the C++ standard

- Early 1970s: C programming language; Dennis Ritchie.
- 1979: Bjarne Stroustrup; 'C with classes'.
- 1983: Name changed to C++.
- 1989:First commercial release.
- 1998: C++98 Standard.
- 2003: C++03 Standard.
- 2011: C++11 Standard.
- 2014: C++14 Standard.
- 2017: C++17 Standard.

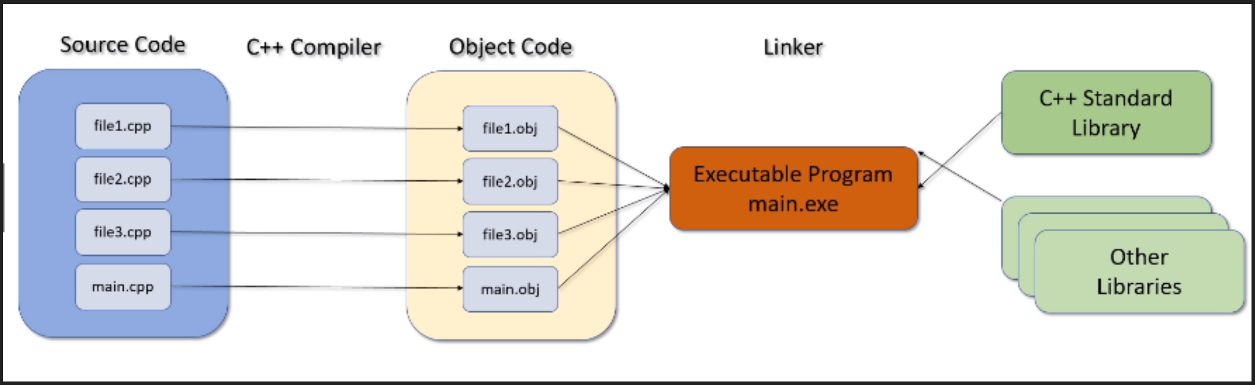
1.2.1 Modern C++ and C++ Standard

- Classical C++: Pre C++11 Standard.
- Modern C++:
  - C++11: Lots of new features.
  - C++14: Smaller changes.
  - C++17: Simplification.

1.3 How does it all work?

- Use non-ambiguous instructions.
- Programming language: source code, high level, for humans.
- Editor: text editor. *.cpp* and *.h* files.
- Binary or other low level representation: object code for computers.
- Compiler: Translates from high-level to low-level.
- Linker: links together our code with other libraries, creates *.exe*.
- Testing and debugging: finding and fixing program errors.

1.3.1 The C++ build process



1.3.2 Integrated Development Environments (IDEs)

- Editor.
- Compiler.
- Linker.
- Debugger.
- Keep everything in sync.

IDEs

- CodeLite.
- Code::Blocks.
- NetBeans.
- Eclipse.
- CLion.
- Dev-C++.
- KDevelop.
- Visual Studio.
- Xcode.

Chapter 2

Installation and setup

2.1 Installing C++ Compiler for windows

- Go to: `http://mingw-w64.org/doku.php/download/mingw-builds`
- Go to: Downloads, find the build, download and run executable.
- Set the environment variable:
  - Control panel → Edit system environment variables.
  - Environment variables → System → Path → Edit.
  - New → Browse → < go to your instaltion dir > → OK.
- Go to CMD: type `c++ --version` → Should print version.

2.2 VSCode Project Setting Up

Inside the .vscode directory add:

- `c_cpp_properties.json`:

```
{
    "configurations": [
        {
            "name": "Win32",
            "includePath": [
                "${workspaceFolder}/**"
            ],
            "defines": [
                "_DEBUG",
                "UNICODE",
                "_UNICODE"
            ],
            "browse": {
                "path": [
                    "${workspaceRoot}",
                    "C:\\Program Files\\mingw-w64\\mingw64\\bin\\gcc.exe"
                ]
            },
            "compilerPath": "C:\\Program Files\\mingw-w64\\mingw64\\bin\\gcc.exe",
            "cStandard": "gnu18",
            "cppStandard": "gnu++14"
        }
    ],
    "version": 4
}
```

- `launch.json`:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "g++.exe - Compilar y depurar el archivo activo",
            "type": "cppdbg",
            "request": "launch",
            "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "miDebuggerPath": "C:\\Program Files\\mingw-w64\\mingw64\\bin\\gdb.exe",
            "setupCommands": [
                {
                    "description": "Habilitar la impresión con sangría para gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "C/C++: g++.exe build active file"
        }
    ]
}
```

- In order to establish the formatting style put the following in `settings.json`:

```
{
    "C_Cpp.clang_format_fallbackStyle": "{ BasedOnStyle: LLVM, UseTab: Never, IndentWidth: 4, TabWi
    "emmet.showSuggestionsAsSnippets": true,
    "files.associations": {
        "*.rmd": "markdown",
        "iostream": "cpp"
    }
}
```

- `task.json`:

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "C/C++: g++.exe build active file",
            "type": "shell",
            "command": "C:\\Program Files\\mingw-w64\\mingw64\\bin\\g++.exe",
            "args": [
                "-g", "main.cpp", "-o", "a.exe" // , "&&", "main"
            ],
            "problemMatcher": [
                "$gcc"
            ],
            "presentation": {
                "echo": false,
                "reveal": "always",
                "focus": false,
                "panel": "shared",
                "showReuseMessage": true,
                "clear": false
            },
            "group": {
                "kind": "build",
                "isDefault": true
            }
        }
    ]
}
```

Chapter 3

Curriculum Overview

3.1 Curriculum overview

Get started quickly programming in C++.

- Getting started.
- Structure of a C++ program.
- Variables and constants.
- Arrays and vectors.
- Strings in C++.
- Expressions, Statements and Operators.
- Statements and operators.
- Determining control flow.
- Functions.
- Pointers and references.
- OPP: Classes and objects.
- Operator overloading.
- Inheritance.
- Polymorphism.
- Smart pointers.
- The Standard Template Library (STL).
- I/O Stream.
- Exception Handling.

Chapter 4

Getting Started

4.1 Writting our first program

- Create a project.
- Create a file and type the following code.
- This program is going to take in a number and then display "Wow that is my favorite number".

```
#include <iostream>
int main() {
    int favorite_number; // stores what the user will enter.
    std::cout << "Enter your favorite number between 1 and 100"; // prints.
    std::cin >> favorite_number;
    std::cout << "Amazing!! That's my favorite number too!" << std::endl; // prints that line. endl adds
}
```

4.2 Building our first program

- Building involves compiling and linking.
- In vscode you can run the compile task by pressing ctrl+b.
- Linking means grabbing all the dependencies the main function needs, making .o or object files and adding them to the executable or the .exe.
- Tipically modern compilers have the option to not produce the object files and go ahead and just produce a single executable. IDEs tipically also hide the object files if they are produced.
- By *cleaning* a project what we mean is the object files are deleted and an executable will be produced.

4.3 What are compiler errors?

- Programming languages have rules.
- Syntax errors: something wrong with the structure.

  ```
  std::cout << "Errors << std::endl; // the string is never terminated.
  ```

- Semantic errors: something wrong with the meaning:

  ```
  a + b; // to sum a and b when it doesn't make sense to add them, maybe they are not numbers for exa
  ```

4.3.1 Examples of errors

Not enclosing a string with the " characters.

```
int main() {
    std::cout << "Hello world << std::endl; // string is not terminated with the other ".
    return 0;
}
```

Typos in your program:

```
int main() {
    std::cout << "Hello world" << std::endll; // endll doesn't exist, this is syntaxis errors.
    return 0;
}
```

Missing semi-colons:

```
int main() {
    std::cout << "Hello world" << std::endl // missing semicolon.
    return 0;
}
```

Function doesn't return the type specified, in this case the function doesn't return an integer.

```
int main() {
    std::cout << "Hello" << std::endl;
    return; // main needs to return an integer and it is returning a void.
}
```

Not returning the specified type.

```
int main() {
    std::cout << "Hello World" << std::endl;
    return "Hello"; // "Hello" is not an integer. Error.
}
```

Missing Curly brace:

```
int main() // opening curly brace missing.
    std::cout << "Hello World" << std::endl;
}
```

Semantic error (example: adding something when it doesn't make sense).

```
int main() {
    std::cout << ("Hello world" / 125) << std::endl; // dividing a string by a number, this doesn't mak
    return 0;
}
```

4.4 What are compiler warnings?

- It is good practice to never ignore compiler warnings.
- The compiler will recognize a potential issue but is still able to produce object code from the source code, things such as uninitialized variables.
- It's only a warning because the compiler is still able to generate correct machine code.
- Example:

```cpp
int miles_driven; // never initialized, this value could be anything.
std::cout << miles_driven << std::endl;
/* Warning: 'miles_driven' is used uninitialized in this function. */
```

- Another example is when you declare variables and never use them.

```cpp
int miles_driven = 100;
std::cout << "Hello world" << std::endl;
/* Warning: unused variable 'miles_driven'. */
```

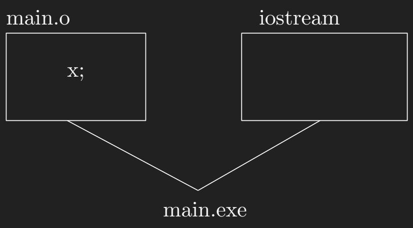
- As a rule you want to produce warning free source code.

4.5 Linked errors

- The linker is having trouble linking all the object files together to create an executable.
- Usually there is a library or object file that is missing.

4.5.1 Example

```cpp
#include <iostream>
extern int x; // this means the variable is defined outside this file.
int main() {
    std::cout << "Hello world" << std::endl;
    std::cout << x;
    return 0;
}
/* This program will compile, but in runtime you will get a linker error. */
```



According to the linker x is undefined, thus an error is produced.

4.6 Runtime Errors

- Errors that occur when the program is executing.
- Some typical runtime errors include:
  - Divide by zero.
  - File not found.
  - Out of memory.
- Can cause your program to crash.
- Exception handling can help deal with runtime errors.

4.7 What are logic errors?

- Errors or bugs in your code that cause your program to run incorrectly.
- Logic errors are mistakes made by the programmer.

Suppose we have a program that determines if a person can vote in an election and you must be 18 years or older to vote.

```
if (age > 18) { // This means that age cannot be 18 thus 18 yearolds would not be able to vote. 18 is no
    std::cout << "Yes you can vote" << endl;
}
```

4.8 Section challenge solution

```
#include <iostream>
int main() {
    int favorite_number;
    std::cout << "Enter your favorite number between 1 and 100: ";
    std::cin >> favorite_number;
    std::cout << "Amazing!! Thats my favorite number too!" << std::endl;
    std::cout << "No really!!, " << favorite_number << " is my favorite number!" << std::endl;
}
/* Output:
Enter your favorite number between 1 and 100: 67
Amazing!! Thats my favorite number too!
No really!!, 67 is my favorite number!
*/
```

Chapter 5

Structure of a C++ program

5.1 Overview of structure of a C++ program

- C++ has lots of keywords.
- Compared to other languages:
  - Java has about 50.
  - C has 32.
  - Python has 33.
  - C++ has 90.
- You can view these words in the following link: c++ reference.
- Keywords cannot be redefined nor used in a way they are not intentionally made for. This is why variables have naming conventions for example.
- Difference between keywords and identifiers, a keyword is something that is built in the programming language, a keyword has specific purpose and meaning that cannot be changed; identifiers are defined and user for the programmers purposes, a variable name is an example of an identifier, another example are function names, there are rules for identifiers, also conventions.
- C++ also has punctiation, and you must adhere to the punctuation rules, things such as "", <<, >>, ;, etcetera, are examples of punctuation.
- When you assemble rules of punctuation, keywords, identifiers, rules, you end up with *syntax* this refers to rules and conventions that you must follow.

5.2 #include Preprocessor directive

- What is a preprocessor?
  - A preprocessor is a program that processes your source code before your compiler sees it. C++ preprocessor strips all the comments from the source code and then sends it to the compiler, it replaces each comment with a single space.
- Directives in C++ start with a # sign. Examples include:

  ```
  #include <iostream>
  #include "myfile.h"
  #if
  #elif
  ```

```
#else
#endif
#ifdef
#ifndef
#define
#undef
#line
#error
#pragma
```

  - In simple terms the compiler replaces the include directive with the file that its refering to, then it recursively processes that file as well.
  - By the time the compiler sees the source code it has been stripped of all comments and all preprocessor directives have been processed and removed.
  - Preprocessor directives are routinely used to conditionally compile code, for example say I want to execute some piece of code only if the program is running in the windows operating system and execute some other if the program is being run by MacOS, preprocessor directives can check to see if you are on windows and run your code, if you're not on windows then the preprocesor directive will run the MacOS code.
  - The C++ preprocessor does not understand C++. It simply processes the preprocessor directives and gets the code ready for the compiler, the compiler is the program that does understand C++.

5.3 Comments in C++

- Comments are programmer readable explanations in the source code; explanations, notes, annotations, or anything that adds meaning to what the program is doing.
- The comments never makes it to the compiler, the preprocessor strips them, the comments are just to enhance the readibility of your source code, it's intended for humans and the compiler never sees it.
- There are basically two ways of writing comments, single line comments and multiline comments.
- Single line comments start with `//`, the remaining characters in that line are considered a comment by the preprocessor.

  ```
  int var = 100; // This is a comment.
  ```

- Multiline comments start with `/*` anything after these characters up until `*/` are considered a comment by the preprocessor. Everything between the `/* */` will be ignored.

  ```
  /* This is a line.
  This is another commented line.
  This is yet another.
  ...
  */ int var = 100;
  ```

- The idea behind comments are to make self documenting code. Self documenting code is the practice of wrinting down what your code does and how it does it so that other people may understand it.

  ```
  int var = 100; // I'm initializing a variable to 100 in order to use <where I will use it> for <sta
  ```

  - Keep in mind not every single line needs commenting, comments must be used only when they are needed, sometimes code can be very hard to read and that's were you want to use comments. You don't want to write a comment for every line of code saying what you do because what you are doing is easily and clearly dedusable, other code however can become very unreadable and complicated and that's were you might want to use comments so that other people know what you are doing. Don't comment the obvious.

  - As a rule: don't comment the obvious, good commenting doesn't justify bad code, and if you made changes to code make sure you also change the comments to save time and confusion, keep comments in sync.

5.4 The main() function

- Every C++ program must have exactly one `main()` function.
- main() is the starting point of program execution.
- the `return 0;` statement in the `main()` function indicates successful program execution.
- When a C++ program executes the operating system calls the `main()` and the code between the curly braces executes. When execution hits the return statement, the program returns the integer 0 to the operating system, if the return value is 0 then the program terminated sucesfully, if the value returned is not 0 then the operating system can check the value returned and determine what when wrong.
- There are two posible ways of writing the main function. They work almost the same and the differences are suttle. All the program actions are contained in the main function.

```
// First way:
int main(/* No parameters. */) {
    // <code>
    return 0;
}
// Second way:
int main(int argc, char *argv[]){
    // <code>
    return 0;
}
```

  - The first version expects no information to be passed in from the operating system in order to run. This is the most common version.
  - The second version allows for parameters to be passed in to the main function when its called, so for example you can pass in an integer or a string that you can then use in your program when it's called from the command line. You would call your program like this in the command line: `program.exe n arg1 arg2 argn` the first argument or `int argc` must be the number of arguments that you will pass in, the next arguments are delimited by spaces and collected and stored inside your program in the `char *argv[]` array which you can use in your program respectively.
  - It's important to see the distinction in order not to get confused when you are looking at code out on the internet.

5.5 Namespaces

- As C++ programs become more complex the combination of our own code, the C++ standard library, and other third party libraries, sooner or later C++ encounters two names repeated, and at that point C++ doesn't know which one to use. This is called a naming conflict and its described when company X named something the same as company Y.
- This is where namespaces come in, you can specify which library you are refering to by using the name of the library and the scope resolution operatior (`::`)
- C++ allows developers to use namespaces as container to group their code entities in to a namespace scope.

- An example is the `std::cout` statement, tecnically it is saying to the C++ compiler to search in "std" or the standard library the function "cout" and use that one.
- Namespaces are introduced to reduce the posibility of a naming conflict.
- However, it can get tedious to write the library name, then the scope resolution operator and finally the function name; thus you can use the `using namespace <insert library name>;` statement to specify that any function from there on will come from the library spefied, now C++ knows which namespace you are using.

  ```
  #include <iostream>
  using namespace std;
  int main() {
      // <code>.
      return 0;
  }
  ```

  – C++ in the code above knows the moment we said `using namespace std;` that any function referenced from that point will be refering to the std library function.

- However there is still a problem, `using namespace std;` doesn't just state to use cout and cin, it brings a lot of other functions we might not know about, for this we can explicitly say we just want to use a certain function of a certain library.

  ```
  #include <iostream>
  using std::cout;
  using std::cin;
  int main() {
      // <code>.
      return 0;
  }
  ```

  – You can still use `cout` and `cin` without having to write `std::cout` and `std::cin` and we are not getting any other names from the standard library of which we won't need. In larger programs its best practice to declare the functions you will use manually and not the namespace.

5.6 Basic input and output