Basic C++ syntax

Raul P. Pelaez

September 7, 2024

Contents

1 Introduction

Let us go through some of the basic syntax in C++ to get you started. You will see most of it is quite similar to C, but with some extensions in some cases and alternatives in others.
Keep in mind that most C code is also valid C++ code, but you should always choose the C++ alternatives whenever possible. Key examples of this are:

- Input/Output (`printf/scanf` in C vs `std::cout/std::cin` in C++)

- Memory handling (`malloc/free` in C vs `std::vector` in C++)

1.1 How to use this document

My intention is for you to look for the source code blocks in this document, and try to compile and run them in your machine. Below the source code blocks, you can find some boxes with additional information, suggestions, and warnings. I encourage you to read them when you encounter something new for you in one of them.

1.2 Trusty online resources

A general word of advice: Be extremely skeptic about online information. Most resources you will find in the wild will be outdated, wrong, or just plain bad. Detecting good resources is a skill that you will develop with time.
In a similar note, be very critical of code generated by LLMs, which have been trained in part with all these terrible resources.

Here are some resources that you can trust:

- `https:\\cppreference.com` is a great reference for the C++ standard library, look here when you want to know more about a function or a type.

- `https:\\godbolt.org` is a great tool to see how your code is compiled, and even compare different compilers and flags. You can also see the assembly code generated by your code.

- `https:\\isocpp.org` is the official site for the C++ standard, you can find the latest standard here, as well as some tutorials and resources. Beware, the standard is very hard to read (looks like a legal document). Keep it in your radar, but it is a very nerdy page.

1.3 Trusty youtube channels

- `https://www.youtube.com/@cppweekly` Weekly uploads about C++, with news, tutorials, and interviews. Tutorials are very focused and short, dealing with very specific concepts each time.

- `https://www.youtube.com/@TheCherno` Has very cool videos on C++ and game development. Take into account many videos were uploaded almost a decade ago, so some things might be outdated. If you are struggling with some fundamental concept, this guy probably has a 15 minutes video explaining it in a very clear way.

- `https://www.youtube.com/@InternetOfBugs` About software development in general, he has only a few videos, which IMO should be mandatory for any software developer to watch. Very down to earth and practical advice.

2 About C++ standards (dialects) and compilation

C++ has evolved a lot since its inception, and the standard has been updated many times. The latest standard is C++20, but most of the code you will find in the wild will be C++11 or C++14 (or even older if you are unlucky). The main reason for this is that the standard is very slow to be adopted, and many compilers are still catching up with the latest features.

We will be working with mostly C++17, stretching into C++20 occasionally, just to get to know the features or to use some the ones that are widely available.

Your compiler will default to a certain dialect unless explicitly instructed otherwise (most probably C++14 or 17). In order to change it you must enable the correct flag, which depends on your compiler.

Remember that we compile a source code called "code.cpp" with:

Source code

```
$ g++ code.cpp -o code # without -o, the default name will be a.out, a.exe or
↪  similar
# Clang++ and g++ are mostly identical
$ clang++ code.cpp -o code
# In windows you might have the MSVC compiler
PS> CL /Fe"path/to/code" code.cpp
```

In order to enforce a particular C++ standard, we add the "std" flag:

Source code

```
$ g++ -std=c++20 code.cpp -o code
PS> CL \std:c++20 code.cpp
```

You might want to enable the "-Wall" flag, which will make the compiler output many warnings when it detects some potentially dangerous code, like an uninitialized variable.

3 Hello world

Our starting point will be a program that prints Hello world, as its usual when learning a new programming language.
Take this quote, which I am paraphrasing, from the C++ creator (Bjarne Stroustrup) as a motivation/consolation:
"It is unheard of that our first code in a new programming language (even something as simple as Hello world) will compile and run first try".
Honestly, I would modify the "our first code" part to "our Nth code". In other words, don't despair, its normal and actually good for your learning that things explode.

Source code

```cpp
#include <iostream> // Makes std::cout and std::endl available
int main(){
  std::cout<<"Hello world"<<std::endl;
  return 0;
}
```

3.1 Key new concepts

Info

iostream
Similar to C, we can import external functionality with the "include" directive. The printing facilities in C come under the header "stdio" (standard input/output), whereas in C++ they are in "iostream" (input/output stream).

Info

Namespaces
C++ introduces the concept of namespaces to avoid name collisions, i.e. what happens if two libraries contain symbols with the same name?. In C, we would avoid naming collisions by prepending our function names with, for instance, the library name, like `library1_foo` and `library2_foo`. In C++ each library can define its symbols in a different namespace, so that we can refer to them as `library1::foo` and `library2::foo`[a]

[a]You can think about namespaces as something similar to how you access symbols of an imported library in Python, say `import numpy as np; np.array`

The construct `std::` can be read as "an entity inside of the 'std' (standard) namespace". Any header (iostream in this case) that we include that its part of the C++ standard library will define all of its symbols under the std namespace.

Suggestion

Try to remove `std::` and compile the code, what do you make of the error?

Info

cout and endl

`cout` takes care of printing stuff to the terminal when fed with something (a string, a number, ...) using the "«" operator.
`endl` is used to end a line. Try to remove it and instead print `"Hello world\n"`.

In small and contained codes such as this, where name collision is unlikely, writing "std::" can be burdensome. Instead we can use the `using namespace`, which allows us to omit it.

Source code

```
#include<iostream> // Makes std::cout and std::endl available
using namespace std; // Every unknown symbol will be assumed to be inside std::
int main(){
  cout<<"Hello world"<<endl;
  return 0;
}
```

4 Input/Output

We know how to print things to the terminal, lets see how to get things `from` the terminal.

Source code

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
  string name;
  cout<<"Enter your name: ";
  cin>>name;
  cout<<"Hello, "<<name<<endl;
  // Types can have methods in C++!
  cout<<"Your name has "<<name.size()<<" letters"<<endl;
  const char *name_c_style_str = name.c_str();
  //Will just print the same string.
  cout<<"C style string: "<<name_c_style_str<<endl;
  return 0;
}
```

4.1 Key new concepts

Info

string
C++ offers this type to handle strings, its like `const char *` in C, but has some benefits, like allowing the possibility or resizing, getting its size, etc.

Info

cin
When we give `cin` a string using the "»" operator and run the program, it will halt the terminal until we press Enter. Whatever we wrote will be stored in the string.

Suggestion

What happens if you write your name with spaces? Try to feed cin with more than one string, or even integers, floats...
Write

Source code

```cpp
float number;
string name, lastname;
cin>>name>>lastname>>number;
```

And write "your_name your_last_name your_age" when prompted. What do the different variables contain?

Info

Types can have methods
In C++, types (like a `struct`, or a `class`) can have methods/functions on top of data. We access the methods using the "." operator, similar to how we access the data members. These methods can even have arguments.

5 Variables

Variables and basic data types are handled in a very similar way to C. We have some new players, like the `auto` keyword.

Source code

```cpp
#include <iostream>
using namespace std;
int main(){
  int a = 1;
  int b = 3;
  int c = a+b;
  cout<<c<<endl; //Prints 4
  // There are many operators available
  c++; // Increments c by 1
  ++c; // Also increments c by 1
  // We can change the value of a variable
  c = 10;
  // We can cast variables to other types
  // Using C-style cast
  double c_double = (double)c;
  cout<<c_double<<endl; //Prints 10.0
  //static_cast is a safer way to cast variables
  float c_float = static_cast<float>(c);
  // There are many types
  double d = 3.1415;
  cout<<"Double has size: "<<sizeof(double)<<" bytes"<<endl;
  float f = 3.14f;
  cout<<"Float has size: "<<sizeof(float)<<" bytes"<<endl;
  char letter = 'a';
  cout<<"Char has size: "<<sizeof(char)<<" bytes"<<endl;
  // We can use auto to let the compiler infer the type
  auto e = 3.14; // e will be a double
  auto g = 'a'; // g will be a char
  // Variables can be marked as constant (inmmutable)
  const int h = 3; // Cannot change after initialization
  // h++; // This will give a compilation error
  // There is a stronger constant type
  constexpr int i = 3; // This is a compile-time constant
  // i++; // This will give a compilation error
  return 0;
}
```

Advanced

C++ offers a library dealing with numeric limits, which can be useful when we want to know the limits of a type. For instance, we can know the maximum value representable by int by writing:

Source code

```
#include <iostream>
#include <limits>
using namespace std;
int main(){
  cout<<"Max int value: "<<numeric_limits<int>::max()<<endl;
  cout<<"Min int value: "<<numeric_limits<int>::min()<<endl;
  return 0;
}
```

The "<>" operator is new in C++, it looks like "less than" or "greater than", but it has absolutely nothing to do with it.
In C++, functions can have two types of arguments, the ones we are used to (like in C) that we pass between parenthesis, and the ones we see here that we pass between "<>". The latter are called "template arguments", and are used to pass types to the function.
Types (structs, class) can have template arguments (we call these types "templated"). In this case numeric_limits is a templated struct, and we are passing the type "int" to it. Finally, we call the member method "max" of these specialized struct.

Info

Transforming strings into numbers
We have two main ways of transforming strings into numbers: 1) A family of free-standing functions from the string header. 2) The `stringstream` class, which is a stream that can be used to read and write from strings.

Source code

```
#include <iostream>
#include <string> // For std::stoi and std::stod
#include <sstream> // For std::stringstream
int main(){
  std::string number_str = "123";
  int number = std::stoi(number_str);
  cout<<"Number: "<<number<<endl;
  number_str = "3.14";
  double pi = std::stod(number_str);
  // Using stringstream
  number_str = "123";
  std::stringstream ss(number_str);
  ss>>number;
  number_str = "3.14";
  ss.str(number_str);
  ss>>pi;
  return 0;
}
```

5.1 Key new concepts

Info

auto
The auto keyword allows the compiler to infer the type of a variable from its initialization. This can be useful when the type is long or complex, or when we want to avoid repeating the type name.

Advanced

See an example of a really complex type. Using the infamous std::chrono library that deals with time:

Source code

```
#include <chrono>
int main(){
  // Without auto (very long and hard to read):
  std::chrono::time_point<
      std::chrono::system_clock,
      std::chrono::duration<long long, std::ratio<1, 1000000000>>>
      longTypeName = std::chrono::system_clock::now();
  // With auto (much cleaner and easier to read):
  auto shortAndClear = std::chrono::system_clock::now();
  return 0;
}
```

6 Scope

Variables have a scope, which is the part of the code where they are visible. In C++, the scope is delimited by curly braces. Variables declared inside a scope are not visible outside of it. Note that the curly braces that delimit a function, a conditional, a loop, etc, are also scopes.

Source code

```
#include <iostream>
using namespace std;
int main(){
  int a = 1;
  {
    int b = 2;
    cout<<a<<endl; // This will print 1
    cout<<b<<endl; // This will print 2
  }
  // cout<<b<<endl; // This will give a compilation error
  return 0;
}
```

Suggestion

There can be no variables with the same name in the same scope.
Try to declare a variable `double a = 3.0;` in the line below the declaration of `int a = 1;`, what happens?
Now try to declare and print the same variable `double a = 3;` in the inner scope, what happens?

7 Conditionals

Conditionals and comparisons are very similar to C. With the addition of some goodies, like the possibility of comparing two strings.

Source code

```cpp
#include <iostream>
#include <string>
using namespace std;
int main(){
  int a = 1;
  int b = 2;
  // Conditional statements
  if (a > b) {
    cout << "a is greater than b" << endl;
  } else if (a == b) {
    cout << "a is equal to b" << endl;
  } else if (a < b) {
    cout << "a is less than b" << endl;
  } else{
    cout << "This will never happen" << endl;
  }
  string name = "Raul";
  string another_name = "not Raul";
  if(name == another_name){
    cout<<"The names are the same"<<endl;
  } else {
    cout<<"The names are different"<<endl;
  }
  // Switch statement
  switch (a) {
  case 1:
    cout << "a is 1" << endl;
    // Break will exit the switch statement
    // Without it, the code will continue executing the next case
    break;
  case 2:
    cout << "a is 2" << endl;
    break;
  default:
    cout << "a is not 1 or 2" << endl;
  }
  return 0;
}
```

Advanced

The Spaceship operator, introduced in C++20, is a three-way comparison operator that contains less than 0, 0, or greater than 0 depending on the comparison result.
This is a very niche operator that can be useful when the types to be compared are not trivial, and the comparison is not trivial either. If computing the comparison is expensive, the spaceship operator can be used to compute it only once.

Source code

```
#include <iostream>
using namespace std;
int main(){
  int a = 1;
  int b = 2;
  // C++20 Spaceship operator
  auto comp = a <=> b;
  if (comp == 0) {
    cout << "a is equal to b" << endl;
  } else if (comp < 0) {
    cout << "a is less than b" << endl;
  } else {
    cout << "a is greater than b" << endl;
  }
  return 0;
}
```

8 Containers

Containers are a very common data structure, and C++ offers a very nice way to handle them with the `std::vector` class. **From now on, you should consider the words "malloc" and "free" as illegal.**
Vector is known as a "contiguous container", which means that all of its elements are stored in a contiguous block of memory. There are other types of containers, like sets or maps, that are not contiguous.

Source code

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(){
  // We can initialize a vector with a list of elements
  vector<int> numbers = {1, 2, 3, 4, 5};
  // We can access elements using the [] operator
  cout<<numbers[0]<<endl; // Prints 1
  // We can add elements to the end of the vector
  numbers.push_back(6);
  // We can remove elements from the end of the vector
  numbers.pop_back();
  // We can remove elements from the middle of the vector
  numbers.erase(numbers.begin() + 2);
  // We can insert elements in the middle of the vector
  numbers.insert(numbers.begin() + 2, 3);
  // We can check the current size of the vector
  cout<<"Size of the vector: "<<numbers.size()<<endl;
  // We can resize the vector
  numbers.resize(10);
  // We can modify individual elements
  numbers[9] = 10;
  return 0;
}
```

Suggestion

See the reference for the std::vector class. I know it looks daunting and inscrutable, this is why its important for you to get used to it.

8.1 Key new concepts

Info

vector
The `std::vector` class is a very versatile container that can hold a contiguous list of elements. It can grow and shrink dynamically, and has many methods to manipulate it.

Info

The <> syntax
This is a template argument, and is used to pass types to a class or function. In this case, we are passing the type "int" to the vector class.
In C, if we want a vector type that works for "int" and "float", we would have to define two different types, i.e. `vector_int` and `vector_float`. In C++, we can define a single type `vector<T>` that can work with any type T (under some rules we will see in the future).

In C you could also make use of Generic templates, but we do not acknowledge the existence of such a thing nor we will talk about it.

Info

Iterators: The `begin` and `end` methods
These methods return an iterator to the beginning and the end of the vector, respectively. We can use these iterators to access the elements of the vector.
You can think of an iterator as a pointer that knows how to move through the elements of a container.
For a vector, the `my_vector.begin()` can be read as "a pointer to the first element of the vector", or in code `&my_vector[0]`.

9 Loops

Loops are also very similar to C, with the addition of the range-based for loop, which is a very nice way to iterate over containers.

9.1 For loops

Source code

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
  vector<int> numbers = {1, 2, 3, 4, 5};
  // Traditional for loop
  for (int i = 0; i < numbers.size(); i++) {
    cout << numbers[i] << endl;
  }
  // Range-based for loop
  for (int n : numbers) {
    cout << n << endl;
  }
  // We can also use auto
  for (auto n : numbers) {
    cout << n << endl;
  }
  // We can also use auto& to get references to each element
  // This is useful when we want to modify the elements
  numbers = {1, 2, 3, 4, 5};
  for (auto& n : numbers) {
    n++; //Will increment each element by 1
  }
  for (auto n : numbers) {
    cout << n << endl;
  }
  return 0;
}
```

9.2 While loops

Source code

```
#include <iostream>
using namespace std;
int main(){
  // While loop
  int a = 0;
  while (a < 10) {
    cout << "a = " << a << endl;
    a++;
  }
  // do while loop
  a = 0;
  do {
    cout << "a = " << a << endl;
    a++;
  } while (a < 10);
  return 0;
}
```

9.3 Key new concepts

Info

Range-based for loop
This is a new way to iterate over containers.

Info

References
The symbol "&" is used to declare a reference. References and pointers are fundamental concepts in C/C++ which we will explore in depth in the future. For now, you can think of a reference as an alias to a variable. When we modify the reference, we are modifying the variable itself.

Source code

```
#include <iostream>
using namespace std;
int main(){
  int a = 1;
  int& b = a;
  b = 2;
  cout<<a<<endl; // Prints 2
}
```