C++ Beginner Notes

Basics of C++ Programming

Muhammad Yousaf

Lecturer CS

5th Oct, 2022

INTRODUCTION

The current ISO C++ standard is officially known as ISO International Standard **ISO C++20.** –Programming Language C++. The older ISO are C++11/C++14/C++17. There's a work in progress for further updates in the version **C++23** by (2023).

What is ISO?

The standard ISO is not intended to teach how to use C++.

It is an international treaty – a formal and legal detailed technical document intended primarily for people writing C++ compilers and standard library implementations.

Things to Understand! Compilers new & old

Not all C++ compilers comply with the latest official ISO C++ standard (C++20) which was finally approved in December 2020.

What does this mean? It means that the code you write, compile and run on one machine may not compile on the next machine if you are using a different compiler (even if it is using the same operating system). In this course you will be using Dev C++ Compiler 2022.

#include <iostream.h> have a .h extension. The C++ standard for this header file is #include <iostream>. Your program may run correctly by adding a .h, but simply adding a .h suffix will not work for all included files. **#include <string.h>** has been replaced with **#include <string>**. The string.h header file does not include C++ strings, it includes C strings.

#include <math.h> has been replaced with **#include <math>**

The lesson to be learnt here is that a lot of C++ code you come across will have been created using older compilers. Some of the header file and other code will not be recognised and may cause compilation errors when compiled on a machine which is more compliant with the C++ standard.

If you are using the Gnu Compiler Collection (gcc, g++ etc) you can specify standards' compliancy via the –std option, e.g., -std=c++17 (other options are C++98, C++11, C++14).

The C++ Standard Template Library (STL)

The C++ programs you write will consist of the classes and functions that you write, or modify (we provide partial solutions to every exercise). However, we will also look at what is still commonly referred to as the STL (now just part of the C++ Standard Library) which holds a collection of existing classes, functions and algorithms that you can use in the programs you will be creating.

Some of the classes and template classes we will be looking at are:

- The **string class**
- The **vector container** template
- The **sort algorithm**
- The **list container** template
- The **find algorithm**

Compiler

Dev C++ 2022

What is Dev C++? It's a freely available integrated development environment (IDE), and comes complete with optional compilers/tools for a variety of languages. We have only installed the IDE and the C++ compiler (cl.exe etc) for this course.

So what is a compiler?

A compiler is a computer program that allows us to write programs in some specially defined language (unfortunately not English, yet) in our case C++ (a so-called **high-level language**). As you heard earlier, the text we write in C++ is called source-code, it is just simple text that one could write in Notepad.exe for example.

Sadly, source-code is not directly understood by operating systems such as Windows, Linux, macOS etc, and it is the compiler's job to convert our **source-code** into the terse, concise and highly optimized **machine-code** that computers understand directly. **Machine-code** is often called **executable or object-code**, and the compiled version of the source code is really just a file that contains numbers – a series of 8-bit bytes, each holding a value between 0 and 255. The numbers either encode low-level instructions (code) or data.

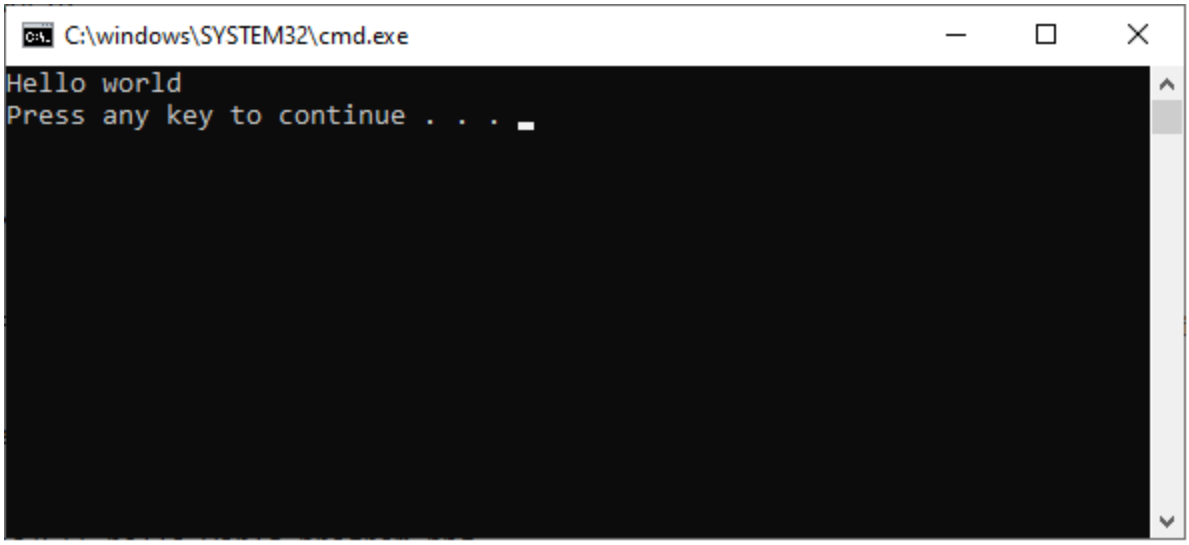
DATA RESULTS

Write down the followings code in your IDE i.e. Dev C++. Build and run the project by pressing the green arrow , or by pressing F5. Output window is known as **Console output**.

```
#include <iostream>

int main()
{
 std::cout << "Hello World!" << std::endl;

 system("pause");
}
```



Explanation

include <iostream>

iostream is a standard C++ library file. Files of this type are called **header files**. They are text files usually containing C++ declarations, definitions, and code.

Including iostream allows us to use some predefined objects to output messages to a console window. This file must be included for any program that outputs data to the screen - or inputs data from the keyboard for that matter.

The C++ compiler operates and passes through our code. In the first pass – called the preprocessing pass

– **#include directives** are processed, which are really text-substitution directives. To be precise, what #include says “Go and find the text file called iostream, and replace the directive with the contents of that file”.

The file name following a #include directive can be enclosed in double quotes or chevrons, i.e., **#include "iostream" or #include** . The meaning of each is defined by the implementation (Microsoft's C++ compiler in our case).

However, it is common practice that if filename is in quotes, it will be looked for locally first - in the same folder as the .cpp file containing it and before being sought elsewhere. Conversely, if it is in chevrons, the compiler will usually look for the file in a central include folder that was installed with the compiler – this folder is usually reserved for include files that come with the compiler, as opposed to those that you might write yourself, e.g., standard includes (like iostream). We therefore usually use quotes when we want to include our own custom, project specific include files, and chevrons when wanting to use include files that come with the compiler.

You may also see that the filename sometimes has a .h file extension. This typically means that the **header** file being used is a C language header file and not a C++ one. There are other variations (you are welcome to use your own), for example, you might also see the occasional .hxx file extension, or something else entirely.

using namespace std;

The **using namespace std**; code above doesn't actually appear in the automatically generated code. In our project and solution files we will add it just after any include directives.

"**namespaces**" in summary, are very simple. Everything within a namespace has a unique name, and so they're often used to help divide larger projects into more manageable chunks and to avoid name clashes.

Standard C++ places a lot of its routines and functionality within a namespace called **std**. We'd like to use this functionality and so we say we want to use all of the std namespace. In the real world, this isn't terribly good practice.

However, by declaring that we're using namespace std throughout, we can access members of this namespace without having to say exactly where they are (in which namespace), or worry about name clashes - where two objects in different files have been given the same name.

The members of std that we will be using throughout are:

- **cin** (Console In), and **cout** (Console Out) and we must use #include <iostream> in order to use them.

- **endl -** is a stream manipulator that writes a new line to output. endl stands for end of line and is pronounced "end all".
- If we don't declare that we were using the **std** namespace (which is the case in the automatically generated code), we have to prefix any use of cin, cout and endl with **std::**, e.g., s**td::cin**, **std::cout**, **std::endl**.
- If you've added the using namespace std; directive, you may now remove the std:: in front of the std::cout << "Hello World!\n"; line.

int main()

This is the entry point to our program – every C++ program must have this same entry point defined. main() is a function, the body of a function is defined within the opening and closing **{ }** braces that come after its name.

The int to the left of main() says that this function returns (or resolves to be/evaluates to) an integer value. Returning a value requires you to use a return statement. However, as it is normal practice to simply return the value zero, if you omit the return statement a value of zero is returned for you (only where main is concerned). If you want to explicitly return the value yourself, you could modify the code to read:

So why return a value anyway? Our added line, **return 0**; means, in this case, that our main, doing whatever it does as part of its function, eventually resolves to the value 0. The value is returned to whatever is called our program's entry point; in this case, and as usually, that is the operating system – which has been waiting for us to return a value at this point. However, the operating system will simply ignore whatever value we ultimately return. Why do it then?

Consider the case when one of your programs needs to run another program to perform a part of its entire function. Your main program needs to start this other program and wait until it completes successfully before continuing. See the description of the system("pause"); code line below.

std::cout << "Hello World!\n";

cout is an object declared in **iostream** – think of its name meaning 'c'onsole 'out'. Anything sent to cout using the double chevron operator << appears on the console the. An alternative to using **\n** is to use endl, e.g., std::cout << "Hello World!" << endl; endl does more or less the same thing as \n.

system("pause");

system() is a standard function that is used to run an external program (here, that's a standard Windows program called pause.exe). The system() function waits for the pause program to complete before continuing. Going back to the discussion on return 0; earlier; this is an example of one program starting another – here that's our program starting the pause program. The pause program returns a value as part of

its completion, and we can examine that if we want. Programs usually return 0 for success, or some other integer value that means something went wrong – the value returned indicates 'what' went wrong. I.e., a returned value of, say, 42 might mean that we can't answer the ultimate question with this computer.

So what is pause? pause.exe is a standard Windows program that simply outputs the message **Press any key to continue . . .** and then waits for a key to be pressed before exiting. We use that facility here so that our program doesn't simply shutdown and disappear as it falls off the end of **main()'**s closing } or encounters a return statement. If we did allow it to exit we would hardly have time to examine our program's output before it disappeared off the screen. Try that and see (comment out the line using a double-slash like this //**system("pause");)** If you run any of the course code on Linux or a Mac, you'll need to change this line to something like **cin.get();.**

Task#01

Task 1 Immediately above system("pause"); Replace the std::cout << "Hello World!\n"; line with the code shown here.

```
#include <iostream>
int main()
{
   int anum = 0;
   std::cout << "Enter a number ";
   std::cin >> anum;
   std::cout << "The number you entered is : "
   << anum
   << std::endl;
   system("pause");
}
```

Test your above program. Tip: remember you may hit the F5 key to build and then run your program.

Try entering some text instead of a number. Did it work? Alter the spacing (use of spaces and tab characters) in your code – does the C++ compiler care about this so called **whitespace**?

Explanation:

The statement int **anum = 0;** defines a variable (a memory location) which we will refer to in our code as anum to store a number of type int to hold an integer (i.e. 1, -4, 27). The memory location is being initialized with a value of **0** here. It is good practice to initialize variables when you define them. In C++11 a new type of uniform initialisation syntax was introduced, and also encouraged, e.g., int anum {0};

cout << "Enter a number "; **cout** is a name that belongs to the namespace **std**. The **<<** is called the **stream insertion operator**. When the program executes, the value to the right of the operator (**"Enter a number"**) is inserted in the output stream and ultimately output to the console window.

std::cin >> anum; cin is the **input stream object** of the namespace **std**. **>>** is called the s**tream extraction operator** and is used here to read a number from the keyboard.

std::cout << "The number you entered is : " << anum << std::endl; This statement concatenates the output by using multiple stream insertion operators. **<<** endl is the stream manipulator that adds a new line.

Header (include) Files

So far we've seen that, to use certain functionality, we had to #include . We have to do similar to use other functionality which is not part of the code C++ language, i.e., we will need to include anything that comes as part of the standard library. In its current form, Visual Studio comes with the following include files (most of which are standard files):

115 C++ Header files:

algorithm	allocators	any	array	atomic
bitset	cassert	ccomplex	cctype	cerrno
cfenv	cfloat	chrono	cinttypes	ciso646
cliext	climits	clocale	cmath	CodeAnalysis
codecvt	complex	condition_variable	csetjmp	csignal
cstdalign	cstdarg	cstdbool	cstddef	cstdint
cstdio	cstdlib	cstring	ctgmath	ctime
cuchar	cvt	cwchar	cwctype	deque
exception	experimental	filesystem	forward_list	fstream
functional	future	hash_map	hash_set	initializer_list
iomanip	ios	iosfwd	iostream	istream
iterator	limits	list	locale	Manifest

map	memory	msclr	mutex	new
numeric	optional	ostream	queue	random
ratio	regex	scoped_allocator	set	shared_mutex
sstream	stack	stdexcept	streambuf	string
string_view	strstream	system_error	thr	thread
tuple	typeindex	typeinfo	type_traits	unordered_map
unordered_set	utility	valarray	variant	vector
xcomplex	xfacet	xfunctional	xhash	xiosbase
xlocale	xlocbuf	xlocinfo	xlocmes	xlocmon
xlocnum	xloctime	xmemory	xmemory0	xstddef
xstring	xtr1common	xtree	xutility	xxatomic