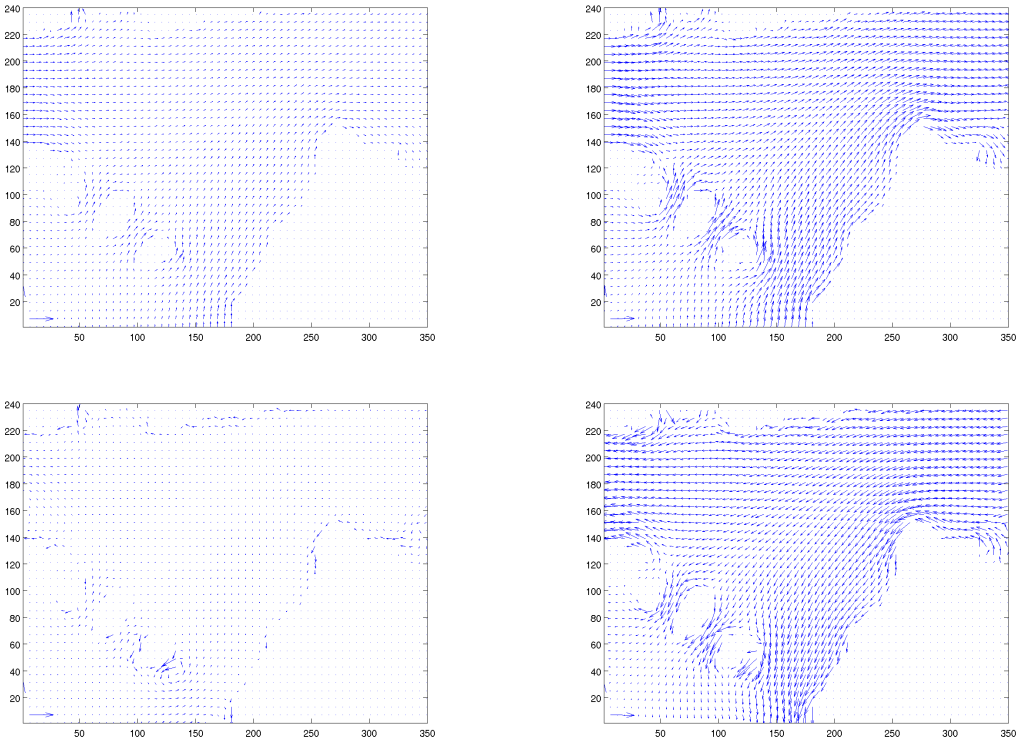
CS319: Scientific Computing

Introduction to CS319

Dr Niall Madden

Week 1: 15+17 January, 2025



Outline

1. Overview of CS319
   - Who we are
   - Classes
   - Class times
   - Materials
   - Text books
   - Assessment
2. CS319: what and why
   - But why C++?
   - Best of both worlds?
3. Scientific Computing
   - Major topics
4. A first example
   - Python
   - Octave/MATLAB
   - C++
5. Introduction to C++
   - Programming Platform
6. Getting started with C++
   - Topics
   - Programming Platform
   - From Python to C++
7. Basic program structure
   - "hello world"

Introduction to C++

Adapted from Wikipedia

C++ (pronounced "C plus plus") high-level, general-purpose programming language created by Bjarne Stroustrup. First released in 1985 as an extension of C programming language, but as an object-oriented language.

In the TIOBE Index for Jan 2025!, C++ is ranked as the 2nd most popular language, behind Python and just ahead of Java, C, C#, JavaScript... It is well ahead of MATLAB (15), and R (18).

But, for context, Scratch is at 12, and Assembly is at 17...

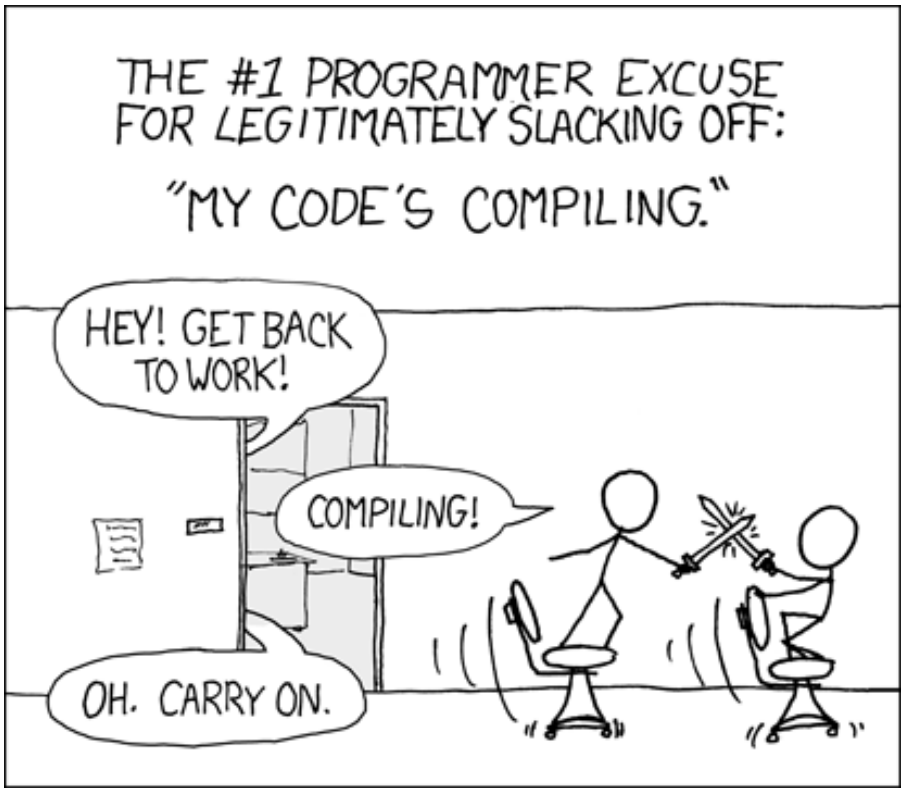
Introduction to C++

The main difference between C++ and (say) Python is the C++ is exclusively a **compiled** language (and not interpreted):

- Write the code
- Compile the code into an executable file.
- Run the executable.

C++ does not have a interactive REPL.

(Don't worry - this will be funny by March).



Source: https://xkcd.com/303

If you have don't have a C/C++ compiler installed on your computer, I suggest using one of

- Code::Blocks
- Bloodshed's Dev-C++
- Xcode (for maxOS)

VS code .

Both are freely available to install on your own device.

For Windows, I suggest installing codeblocks-20.03mingw-setup.exe, since this includes compilers as well as the IDE.

To get started, try an online compiler such as https://www.onlinegdb.com or http://cpp.sh

The C++ topics we'll cover are

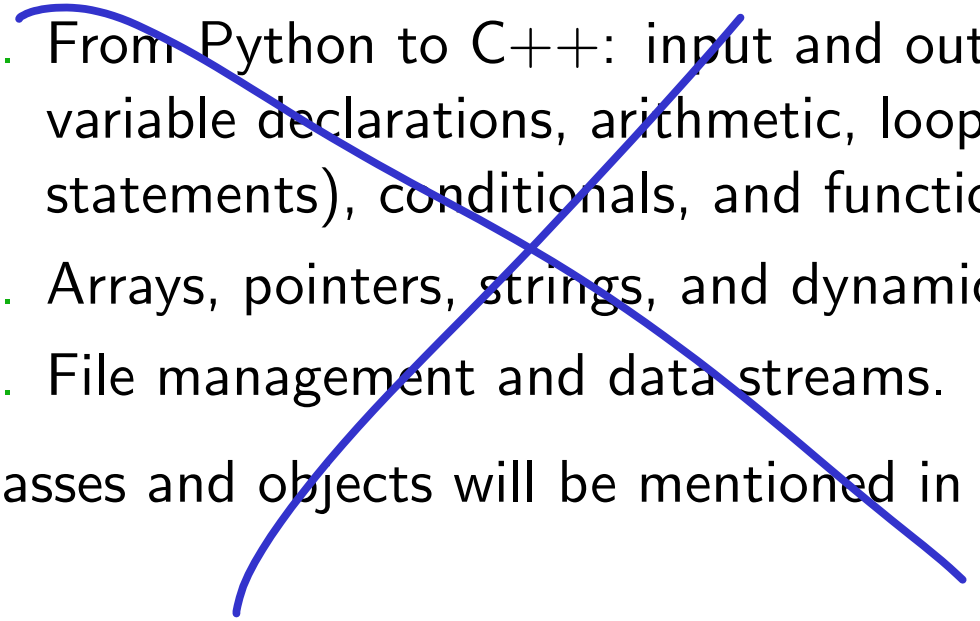
1. From Python to C++: input and output, data types and variable declarations, arithmetic,
2. loops, Flow of control (`if` statements), conditionals,
3. functions.
4. Arrays, pointers, strings, and dynamic memory allocation.
5. File management and data streams.
6. Introduction to classes and objects.

Other topics, eg STL.

The C++ topics we'll cover are

1. From Python to C++: input and output, data types and variable declarations, arithmetic, loops, Flow of control (`if` statements), conditionals, and functions.
2. Arrays, pointers, strings, and dynamic memory allocation.
3. File management and data streams.

(Classes and objects will be mentioned in passing).



To get started, we'll use an online C++ compiler. Try one of the following

- https://www.onlinegdb.com
- http://cpp.sh
- https://www.programiz.com/cpp-programming/online-compiler/

Later (once it is properly installed) we can use a C++ compiler and IDE that is installed on the PCs in lab. Most likely, this will be Code::blocks.

On your own device, try installing one of the following free IDE's and compilers.

- Windows: `Code::blocks` (install `codeblocks-20.03mingw-setup.exe`)
- Windows: Bloodshed's `Dev-C++`
- maxOs: Xcode
- Linux: it is probably already installed!

The convention is the give C++ programs the suffix `.cpp`, e.g., `hello.cpp`. Other valid extensions are `.C`, `.cc`, `.cxx`, and `.c++`.

If compiling on the command line with, e.g., the GNU Project's C/C++ compiler, the invocation is

```
$ g++ hello.cpp
```

or `clang hello.cpp`.

If there is no error in the code, an executable file called `a.out` is created.

The workflow is different with an IDE: we'll demo that as needed.

Most/all of you have some familiarity with Python. There are numerous resources that introduce C++ to Python-proficient programmers.

For example: https://runestone.academy/ns/books/published/cpp4python/index.html One of its advantages is that it allows you to try some code in a browser.

Let me know if you find any other useful resource.

Basic program structure

- A "header file" is used to provide an interface to standard libraries. For example, the *iostream* header introduces I/O facilities. Every program that we will write will include the line:

```
#include <iostream>
```

Python Comparison

This is a *little* like `import` in Python.

- Like Python, the C++ language is case-sensitive. E.g., the functions `main()` and `Main()` are not the same.

Basic program structure

- The heart of the program is the `main()` function – every program needs one. When a compiled C++ program is run, the `main()` function is run first. If it is not there, nothing happens!
- "Curly brackets" are used to delimit a program block.

Python Comparison

This is similar to the use of "*colon and indentation*" in Python.

Example:

```
if (a == b):
    print("a=b")
```

```
if (a == b)
{
    std::cout << "a = b";
}
```

Basic program structure

- Every (logical) line is terminated by a semicolon;
  Lines of code not terminated by a semicolon
  are assumed to be continued on the next line;

- Two forward-slashes // indicate a comment – everything after them is ignored until an end-of-line is reached.

Python Comparison

So, this is similar to a # in Python.

This program will output a single line of output:

00hello.cpp

```
#include <iostream>
int main()
{
  std::cout << "Howya␣World.\n";
  return(0);
}
```

using namespace std;

00hello.cpp

```
#include <iostream>
int main()
{
  std::cout << "Howya␣World.\n";
  return(0);
}
```

- the identifier `cout` is the name of the **Standard Output Stream** – usually the terminal window. In the programme above, it is prefixed by `std::` because it belongs to the *standard namespace*...
- The operator `<<` is the **put to** operator and sends the text to the *Standard Output Stream*.
- As we will see `<<` can be used on several times on one lines. E.g. `std::cout << "Howya World." << std::endl;`

Variables

Variables are used to temporarily store values (numerical, text, etc, ...) and refer to them by name, rather than value.
Unlike Python, all variables must be declared before begin used.
Their **scope** is from the point they are declared to the end of the function.

More formally, the variable's name is an example of an **identifier**.
It must start with a letter or an underscore, and may contain only letters, digits and underscores.

Examples: Ok: var_name; _start; name123;
ThisIs_fine_999_too;

WeCanAlsoHaveVeryLongVariableNames;

Not OK: 3start_with_digit has-a-dash c++

Variables

All variables must be defined before they can be used. That means, we need to tell the compiler the variable's name and **type**. Every variable should have a **type**; this tells use what sort of value will be stored in it. The type does not change (usually).

Python comparison

In Python, one "declares" a variable just by using it. The type of the variable is automatically determined. Furthermore, its type can change when we change the value stored in the Python variable.
This is one of the reasons why Python is so flexible, and so slow.

Variables

The variables/data types we can define include

- `int` → integer, a positive or negative whole number
- `float` → number with decimal place (roughly 8 digits).
- `double` → number with decimal place and about 16 digits.
- `char` → a single character, eg 'a', 'z', ...
- `bool` → True or False

Finished here Friday