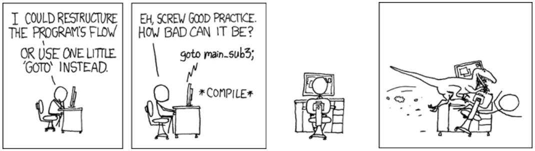
Annotated slides from 9am class

CS319: Scientific Computing

Getting Started with C++

Dr Niall Madden

Week 2: **9am and 4pm**, 17 January, 2024


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

Source: xkcd (292)

Class times

	Mon	Tue	Wed	Thu	Fri
9 – 10			✓	LAB(?)	
10 – 11					
11 – 12					LAB(?)
12 – 1					LAB(?)
1 – 2					
2 – 3					
3 – 4					
4 – 5			✓		

- My thanks to those who sent me your time-table information.
- Based on that everyone can attend at least two of
  - Thursday 9-10
  - Friday 11-12
  - Friday 12-1.
- First lab is next week (Week 3).
- **Any questions?**

Outline

Getting started with C++

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!

IDE = Integrated Development Environment

Getting started with C++ Programming Platform

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
```

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 Python program file names in in ".py".
For c++ it is ".cpp"

From Python to C++

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.
- 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!

Basic program structure

- "Curly brackets" are used to delimit a program block.

Python Comparison

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

Example: Python

```
if (a == 10):
    print("a is 10")
    print("this is displayed if a is 10")
```

not run if a ≠ 10

C++

```
if (a ==10 )
{
    cout << "a is 10";
    cout << "another line";
}
```

If we leave out the { and } only the first line is part of the block.

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.

For multiline comments:
Python: """ three quotes indicate a multiline comment (or doc string) """
C++: /* this is a comment */

```
/* And so
is this
piece of
text */
```

Basic program structure

This program will output a single line of output:

00hello.cpp

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

The \n means "new line". One could also write

std::cout << "howya world" << std::endl;

"endl" = "End Line"

Basic program structure

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(or block).

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:

Valid identifiers: __hello ThisIsAnIdentifier Name12345 _and_so_is_this_ x x1 x1x1x1x

Invalid identifiers: 1name (starts with a digit)
ThisIsNo tOk (contains a space)
--hello ("minus", or any other sympol other than _, not allowed).

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 things that makes Python so flexible, and so slow.

Variables

The variables/data types we can define include

- `int`
- `float`
- `double`
- `char`
- `bool`

Finished here at 9.50