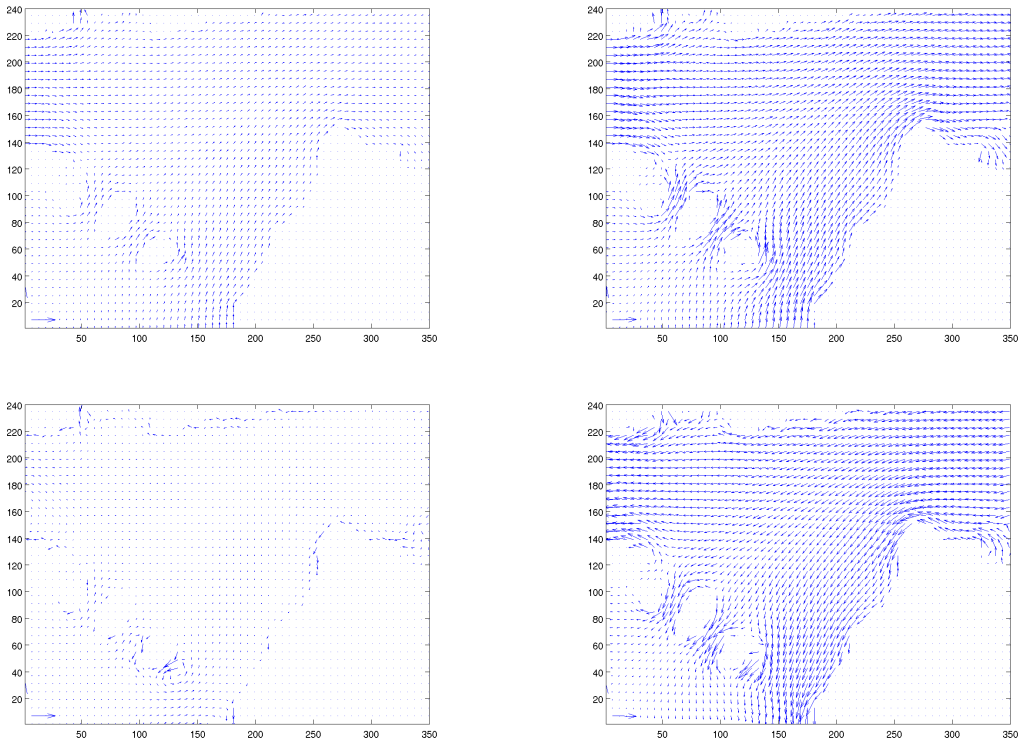
CS319: Scientific Computing

Introduction to CS319

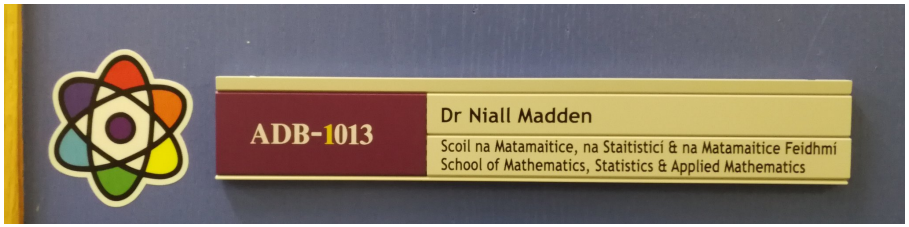
Dr Niall Madden

Week 1: 10 January, 2024



Outline

Lecturer details:



Who: Dr Niall Madden (he/him)
How to greet: Niall ("Knee-al" `#StartsWithAName`)
From: School Mathematical and Statistical Sciences.
Where: Office: AdB-1013, Arás de Brún.
Contact: Email: Niall.Madden@UniversityOfGalway.ie

Students: 3rd year Mathematics and/or Computing (3BS9); 3nd and 4th year Mathematical Science (3BMS2+4BMS2), 4th Year Maths/Applied Maths (4BS2); 4th Data Science, MSc Mathematics; Visiting Students (??).

This module involves a mix of lecture "classes" and lab sessions.

- The "classes" will mix conventional lectures, and practical sessions. That is, you will spend some time coding in every class.
- All slides and lecture materials (such as sample programmes) will be made available in before the start of the 9am class.

We'll use Canvas for

- Posting announcements (1 per week, usually);
- Posting grades
- Assignment uploads
- *Links* to slides and scripts from class.
  All materials will actually be hosted at
  `https://www.niallmadden.ie/2324-CS319`

Code (and there will be lots of it) will be available on a `git` repository at: `https://github.com/niallmadden/2324-CS319`
Later I'll send you an invitation to it. If you didn't get it, send me an email.

The notes for CS319 are largely self-contained. But some books will be VERY helpful. The reading list is at https://nuigalway.rl.talis.com/modules/cs319.html but will be updated. Key books include

- Scientific Computing with Case Studies (Diane O'Leary) https://epubs-siam-org.nuigalway.idm.oclc.org/doi/book/10.1137/9780898717723
- Practical C++ programming (Steve Oualline). https://search.library.nuigalway.ie/permalink/f/1pmb9lf/353GAL_ALMA_DS515666310000362 6
- Think Python (Allen B. Downey) https://greenteapress.com/wp/think-python-2e/
- Learning MATLAB (Toby Driscoll). https://epubs-siam-org.nuigalway.idm.oclc.org/doi/book/10.1137/1.9780898717662
- More will be added ...

The final grade for CS319 will be based on

- **Four programming assignments** (40%)
- a mid-semester open-book test (20%) (Week 7).
- a project and presentation (40%)

This module does not have an end-of-semester exam.

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

We need to find some other times for labs.

Please send your time-table to Niall (Niall.Madden@UniversityOfGalway.ie) ASAP, preferably today.

CS319: what and why

In CS319 we are primarily concerned with *three* issues:

1. How to use a computer to solve a scientific problem. That is:
   - how to determine the best algorithm to apply in a given situation.
   - how to understand the potential and limitations of the algorithm.
2. Implementing that algorithm: How to write the code! (But in what language??)
3. Testing/verifying/validating the implementation.

CS319: what and why

More deeply, this is a course on **programming and problem-solving**.

It is NOT a "first course on programming". You are expected to be proficient in at least one language. For most of you, that language is Python. For some it is Java. (Any others?)

The primary learning outcome is that, by the end of the semester, you can honestly list "Skilled in scientific computing" on your CV.

We'll discuss 3 candidates for a language with which to do Scientific Computing:

- C/C++
- MATLAB/Octave
- Python

Scientific Computing

Dianne O'Leary describes a **computational scientist** as someone whose focus is the intelligent development and use of software to analyse mathematical models.

These models arise from problems formulated by scientists and engineering. Solutions/models can then be constructed using statistics and mathematics. Numerical methods are then employed to design algorithms for extracting useful information from the models.

Scientific Computing

In scientific computing, we are interested in the **correct**, **reliable** and **efficient** implementation of these algorithms. This requires knowledge of how computers work, and particularly how numbers are represented and stored.

History has shown that mistakes can be very, very costly.

Source: Wikipedia

For us, the major topics of CS319 will be

- **Computer representation of numbers**, as well as more complicated objects, such as vectors and matrices.
- Visualisation of functions and data
- **Root-finding and optimisation**
- Data fitting, and least squares;
- Efficiency and complexity of algorithms (from an experimental/applied view).
- Solving linear systems by direct and iterative methods;
- Representation and visualisation of graphs and networks

A first example

In the first few weeks of the module, we'll use C++.
But first we are going to study, without too much explanation, how to implement a simple algorithm in each of these three languages, and estimate their (in)efficiency.
The problem is to write some code that will sum all the elements in a list, and report how long it took.
In each case, we'll take the simplest possible approach, and ignore that each of these languages has (somewhat built-in) functions to do this.

TimeAlg1.py

```
# Sum the elements of a list in Python
import time

N=10**7       # N=10^n
A = [1]*N
start = time.time()
s1=0;
for i in range(len(A)):
    s1+=A[i]
t1 = time.time() - start
print(f"N={N:6.0e}, error={s1-N}, time(s)={t1:6.2f}")
```

TimeAlg1.m

```
% Sum the elements of a link in MATLAB/Octave
N = 10^6;    % N=10^n
A = ones(1,N);
start=tic;
s1 = 0;
for i=1:length(A)
   s1=s1+A(i);
end
t1=toc(start);
fprintf('N=%8.2e, error=%d, time(s)=%8.4f\n',...
         N, s1-N, t1)
```

TimeAlg1.cpp

```
#include <iostream>
#include <time.h>
#include <math.h>
int main() {
  int N=pow(10,6); // N=10^n
  double *A = new double [N];
  for (int i=0; i<N; i++)
    A[i]=1.0;
  clock_t start=clock();
  double s1=0;
  for (int i=0; i<N; i++)
    s1+=A[i];
  double num_clocks = (double)(clock()-start);
  double t1 = num_clocks/CLOCKS_PER_SEC;
  std::cout << "N=10^" << log10(N)
            << ", error=" << s1-N
            << ", time(s)" << t1 << std::endl;
  return(0);
}
```

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 2024, C++ is ranked as the 3rd most popular language, behind (in order) Python, and C, and just ahead of Java and C#.

Introduction to C++

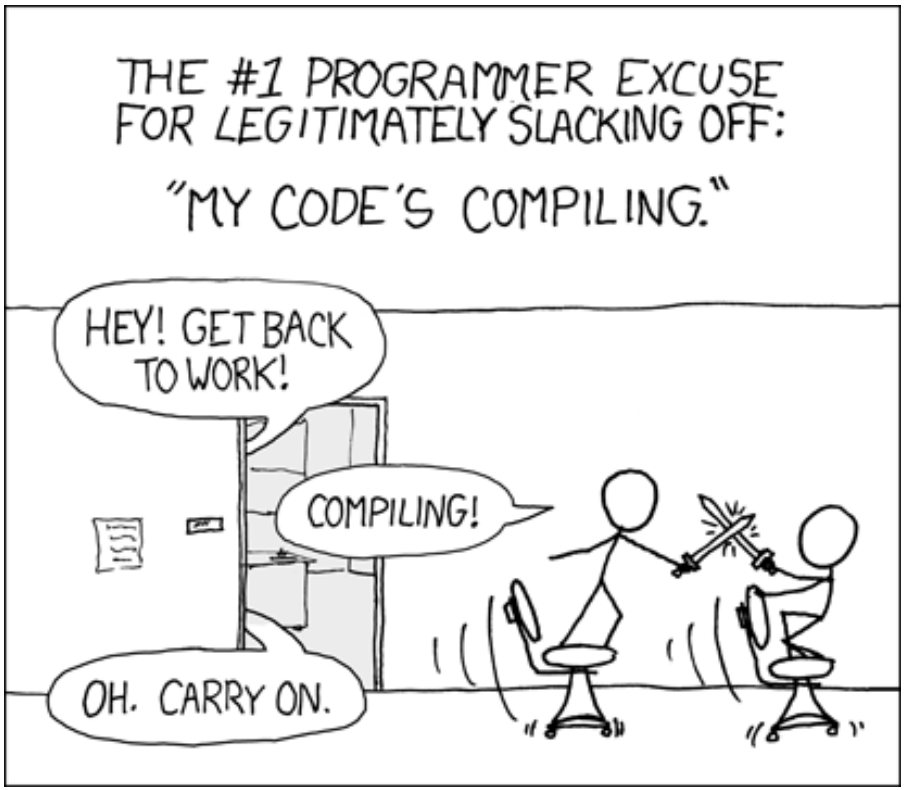
The main difference between C++ and (say) Python is the C++ is exclusively a **compiled** language (and not interpreted):

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

C++ does not have a interactive REPL.

Introduction to C++

(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)

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, 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).

C++ is a programming language developed as an extension to C. It is a *superset* of C, so C programs can be compiled on a C++ compiler.
The convention is the give C++ programs the suffix `.cpp`, e.g., `hello.cpp`. Other valid extensions are `.C`, `.cc`, `.cxx`, and `.c++`.

The GNU projects C/C++ compiler is used. On the command line, the invocation is

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

If there is no error in the code, an executable file called `a.out` is created.
If the present working directory is contained in your PATH variable, just type its name to run it. Otherwise, indicate that it is in your PWD: `./a.out`

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

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

- Unlike Python, all variables must be declared before begin used. However the definitions do not have to be grouped together at the start of a function – they can occur at any point in the function. Their **scope** is from the point they are declared to the end of the function.

- The heart of the program is the `main()` function – every program needs one. `void` is the default argument list and can be omitted.

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

- "Curly brackets" are used to delimit a program block. This is similar to the use of indentation in Python.

- 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. So, this is similar to a # in Python.

To output a line of text in C++:

```
#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." << "\n";
```

Variables

Variables are used to temporarily store values (numerical, text, etc, ...) and refer to them by name, rather than value.
More formally, the variable's name is called the **identifier**. It must start with a letter or an underscore, and may contain only letters, digits and underscores.

Examples:

Variables

All variables must be defined before they can be used. That means, we need to tell the compiler before we use them. This can be done at any stage in the code, up to when the variable is first used.

Every variable should have a **type**; this tells use what sort of value will be stored in it.

The variables/data types we can define include

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

Variables

Integers (positive or negative whole numbers), e.g.,

```
int i; i=-1
int j=122;
int k = j+i;
```

Floats These are not whole numbers. They usually have a decimal places. E.g,

```
float pi=3.1415;
```

Note that one can initialize (i.e., assign a value to the variable for the first time) at the time of definition. We'll return to the exact definition of a `float` and `double` later.

Variables

Characters Single alphabetic or numeric symbols, are defined using the `char` keyword:

`char c;` or `char s='7';`

Note that again we can choose to initialize the character at time of definition. Also, the character should be enclosed by single quotes.

Arrays We can declare **arrays** or **vectors** as follows:

```
int Fib[10];
```

This declares a integer array called `Fib`. To access the first element, we refer to `Fib[0]`, to access the second: `Fib[1]`, and to refer to the last entry: `Fib[9]`.

As in Python, all vectors in C++ are indexed from `0`.

Variables

Here is a list of common data types. Size is measured in bytes.

Type	**Description**	(*min*) **Size**
`char`	character	1
`int`	integer	4
`float`	floating point number	4
`double`	16 digit (approx) float	8
`bool`	`true` or `false`	1

See also: `02variables.cpp`

In C++ there is a distinction between **declaration** and **assignment**, but they can be combined.

As noted above, a `char` is a fundamental data type used to store as single character. To store a word, or line of text, we can use either an *array of* `chars`, or a `string`.
If we've included the `string` header file, then we can declare one as in: `string message="Well, hello again."`
This declares a variable called `message` which can contain a string of characters.

03stringhello.cpp

```
#include <iostream>
#include <string>
int main()
{
  std::string message="Well, hello again.";

  std::cout << message << "\n";
  return(0);
}
```

In previous examples, our programmes included the line
`#include <iostream>`
Further more, the objects it defined were global in scope, and not exclusively belonging to the *std* namespace...

A **namespace** is a declarative region that localises the names of identifiers, etc., to avoid name collision. One can include the line
`using namespace std;`
to avoid having to use `std::`