Introduction to C++: Workshop One

Dr. Alexander Hill

a.d.hill@liverpool.ac.uk

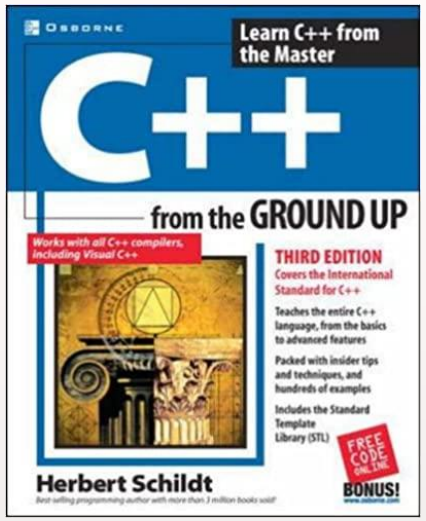
October 2024

LIV.INNO

Course Aims

- Introduce you to the C++ programming language
- Run through the syntax and basic operations
- Work through some examples together, increase fluency
- Introduction to Monte Carlo methods
- Complete a collaborative project using C++ with a Monte Carlo context

Resources



- https://alex-hill94.github.io/#teaching
- C++ from the ground up, Herbert Schildt
- Online compiler: https://www.programiz.com/cpp-programming/online-compiler/
- Online tutorials: https://www.w3schools.com/cpp
- Interview with C++ creator - https://www.youtube.com/watch?v=uTxRF5ag27A
- Jan Kretzschmar (jan.kretzschmar@liverpool.ac.uk)
- LLMs (e.g. ChatGPT, Claude) to research concepts, I would highly recommend working through problems yourself so that you get a better personal feel for the language

Course Composition

- Weekly two-hour workshops up until the 21st of November
- Homework to be done in own time
- Group project and presentation

Aim of Workshop One

- Introductions
- History and philosophy of C++
- Get set up with a text editor and compiler
- Run a script

Introductions

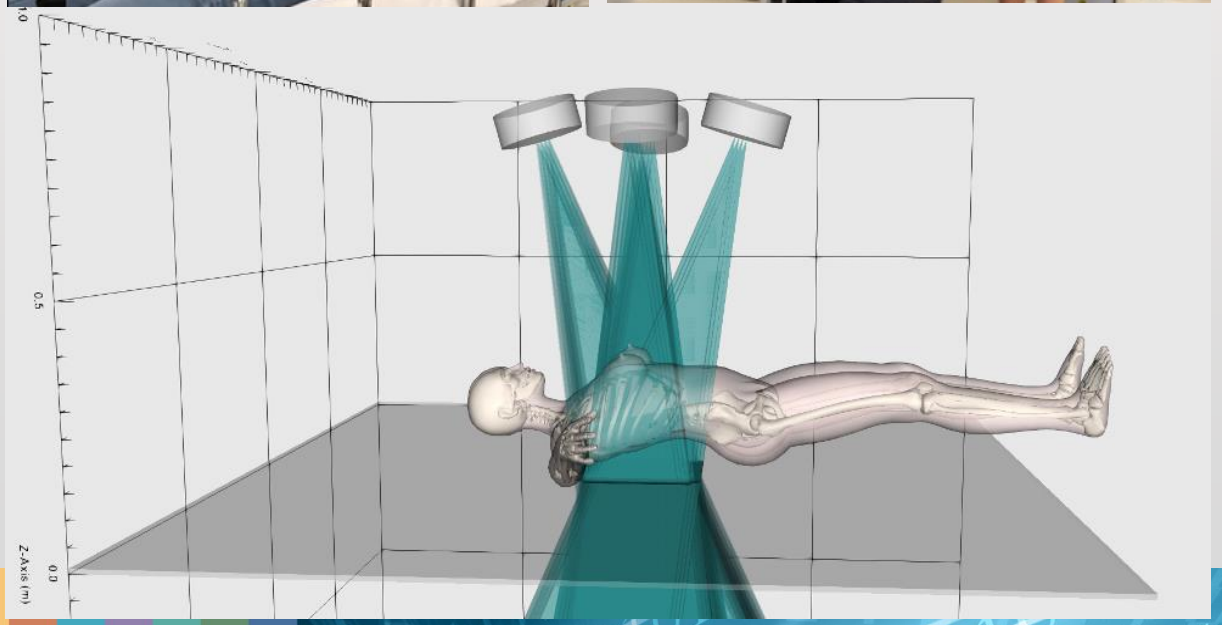
- Started PhD in astrophysics in 2017 at the ARI
- Part of the first LIV.DAT cohort (precursor to LIV.INNO)
- Researched cosmological simulations, now working on medical physics
- Primarily use Python

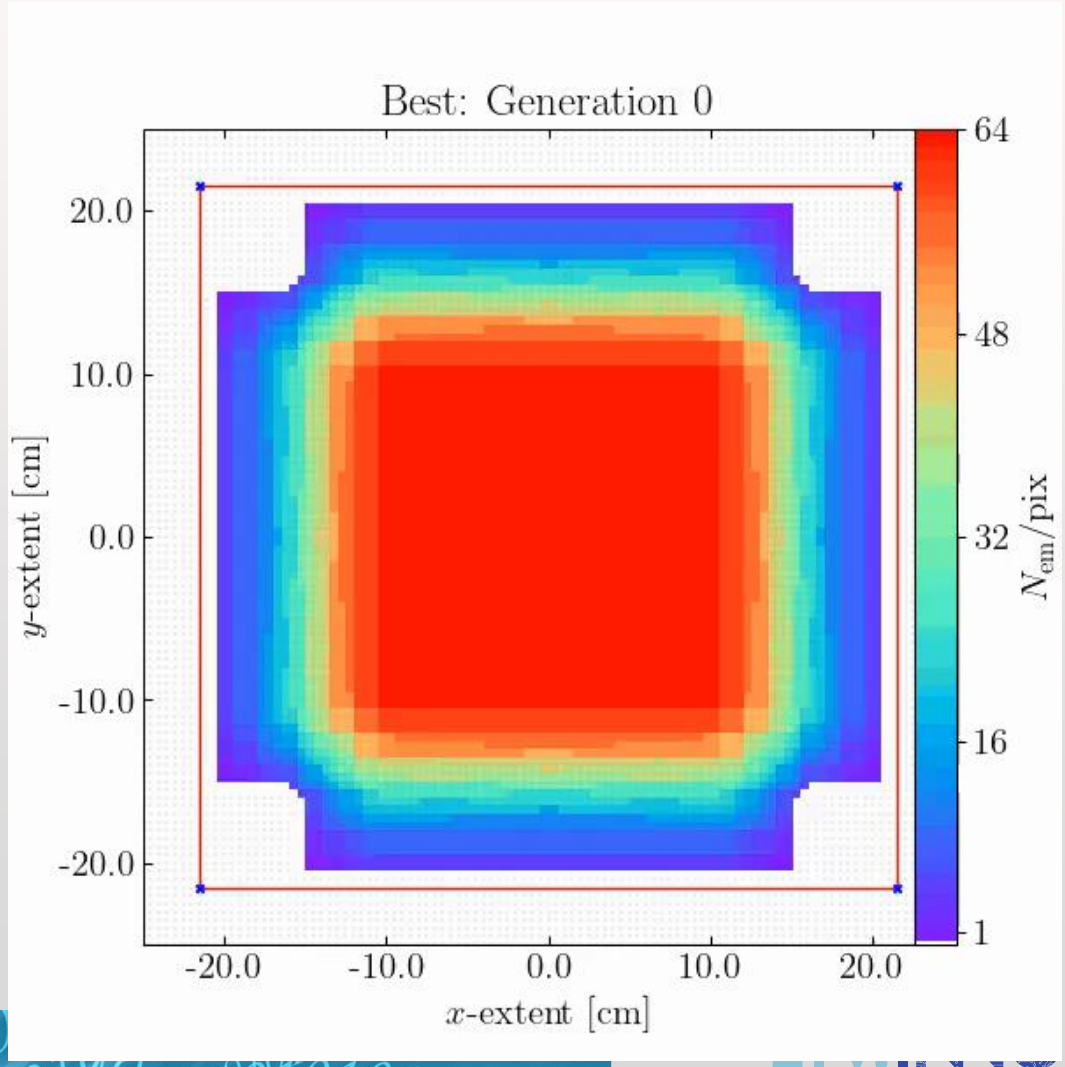
The EAGLE simulations

EVOLUTION AND ASSEMBLY OF GALAXIES AND THEIR ENVIRONMENTS

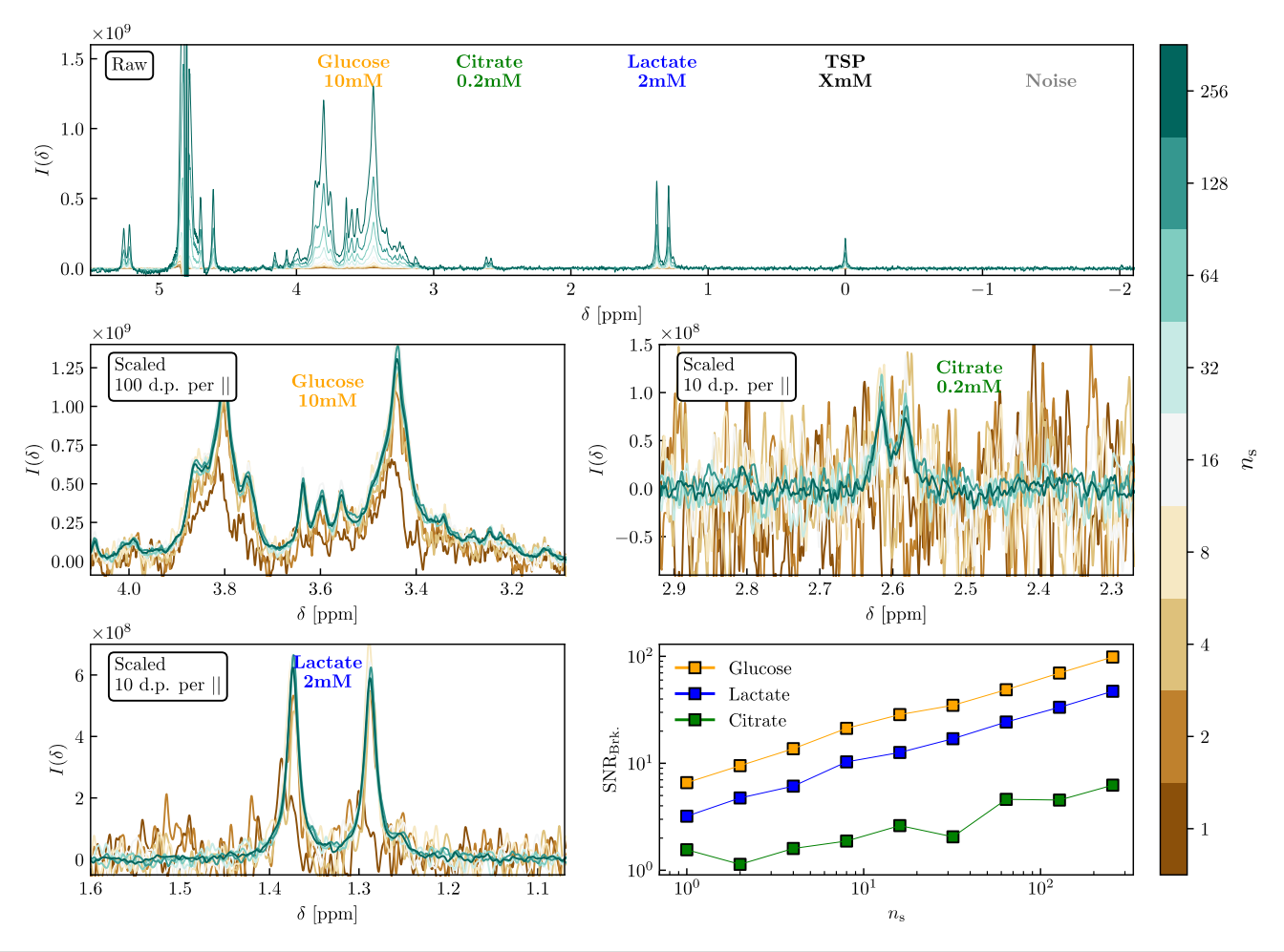
A project of the Virgo consortium

Current research: medical physics





Current research: medical physics



My role here

- Conduct my own research, co-supervise several students
- Connect with industry
- Lead some training and organise data science seminar series
- Help you! Ask me for advice with coding, placements, paper writing, living in Liverpool, etc.

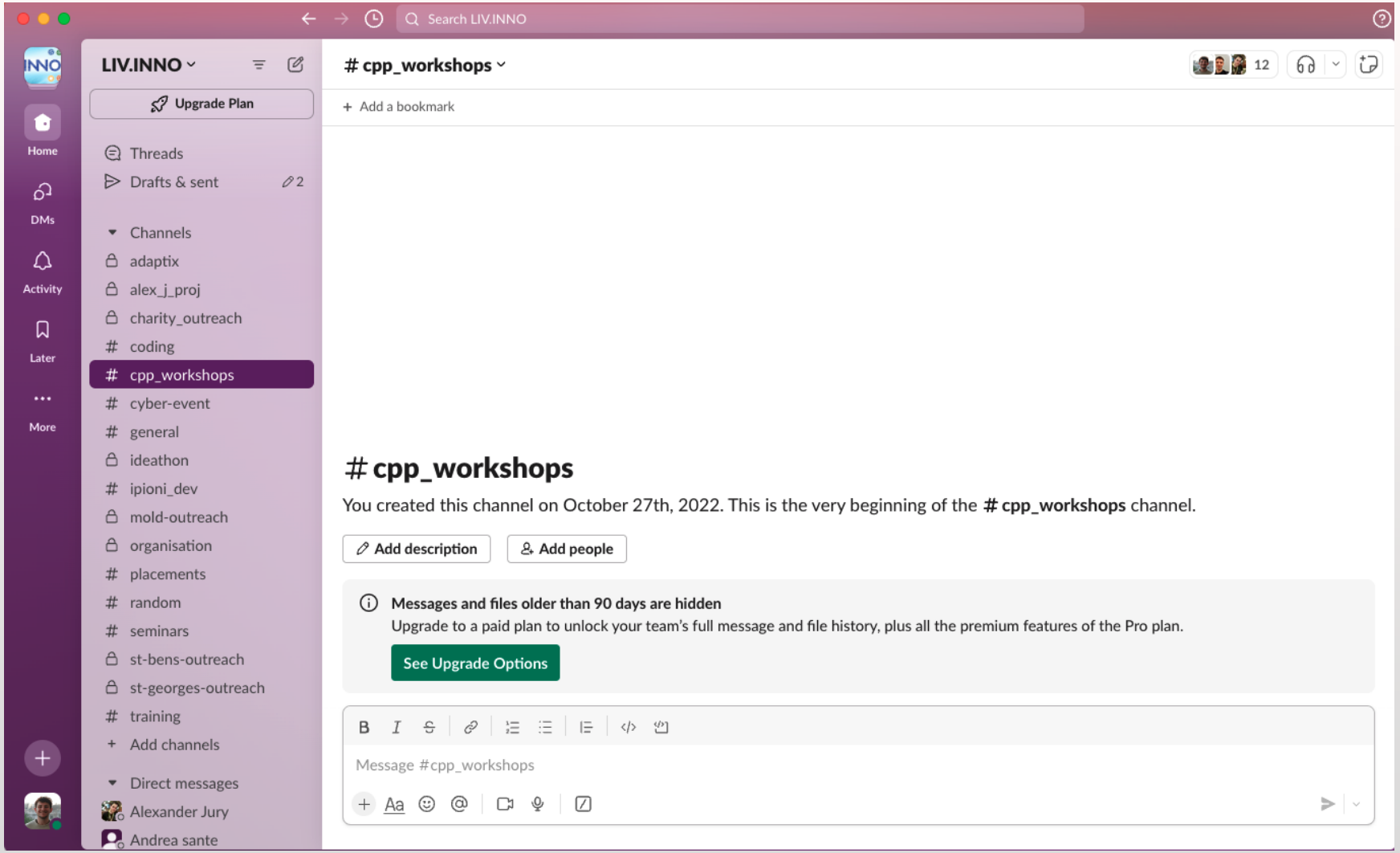
Introductions

Introduce yourself! (background, coding experience, research interests)

What do you want to get from this PhD, and this course in particular?

What coding challenges might you have over the course of your PhD?

Slack channel



https://join.slack.com/t/livinno/shared_invite/zt-2syibrx4g-lr8xu7_9Q_sGM6vGlJhoAA

What is C++?

What is C++?

- C++ is a superset of the programming language C (so two languages for the price of one!)
- Embodies the philosophy of Object-Oriented Programming
- Extended set of libraries
- Millions of developers worldwide
- Commonly used in conjunction with languages like Python on big projects

What is C?

What is C?

- Created in the 1970s by Dennis Ritchie and Ken Thompson at Bell Labs
- The first ‘programmer’s language’: general purpose and human-readable
- A middle-level programming language

Programming languages

Low level

- E.g. Assembly language, machine code
- Provides nothing more than direct access to the computer hardware
- Requires explicit memory management
- Little (if any) abstraction
- Maximum control

Middle level

- E.g. C, C++
- Provides a user with a concise set of tools, while still offering flexibility with data management
- Balances control and ease of use

High level

- E.g. Python, Perl,
- Aims to give the programmer everything they could want
- Highly human-readable and abstract
- Automatic memory management
- Sacrifices some performances for ease of use

```asm
; x86 Assembly (NASM syntax) - Low Level
; Hello World
section .data
    msg db 'Hello, World!',0xA
    len equ $ - msg

section .text
global _start
_start:
    mov eax, 4      ; sys_write
    mov ebx, 1      ; stdout
    mov ecx, msg    ; message
    mov edx, len    ; length
    int 0x80        ; kernel interrupt

    ; Adding two numbers (3 + 4)
    mov eax, 3      ; First number
    mov ebx, 4      ; Second number
    add eax, ebx    ; Result stored in eax

    mov eax, 1      ; sys_exit
    mov ebx, 0      ; exit code
    int 0x80
```

```c
// C Programming - Mid Level
#include <stdio.h>

int main() {
// Hello World
printf("Hello, World!\n");
// Adding two numbers
int a = 3;
int b = 4;
int sum = a + b;
printf("Sum: %d\n", sum);
return 0;
}
```

```python
# Python - High Level
# Hello World
print("Hello, World!")

# Adding two numbers
a = 3
b = 4
sum = a + b
print(f"Sum: {sum}")
```

Features of C

- C allows you to manipulate the constituent components of your computer
- No buffer between programme and hardware
- Requires the user to define routines for performing high-level operations
- Manual memory management required
- Great for situations where performance is critical
  - Did you know that most of NumPy is written in C (with some C++)?

Why C++?

- A better way to manage greater complexity
- As the required tasks of computers become more complex, a higher level of abstraction is required
- Object oriented programming is a way to achieve this
- Little sacrifice in the efficiency and flexibility of C
- Backwards compatible with C

C++ History

- Created by Bjarne Stroustrup in 1979 at Bell Labs
- Originally called ‘C with Classes’
- Renamed C++ due to the ++ increment operator (a = 0, a++, print(a) => 1)

Procedural programming

- Programmes run step-by-step in a logical order
- Code is comprised of data and the functions which act on them
- Each function does one thing

Object oriented programming

- Idea: decompose a problem into constituent parts (components)
- Each component has its own rules and data
- Complexity is reduced
- Three characteristics: encapsulation, polymorphism, and inheritance

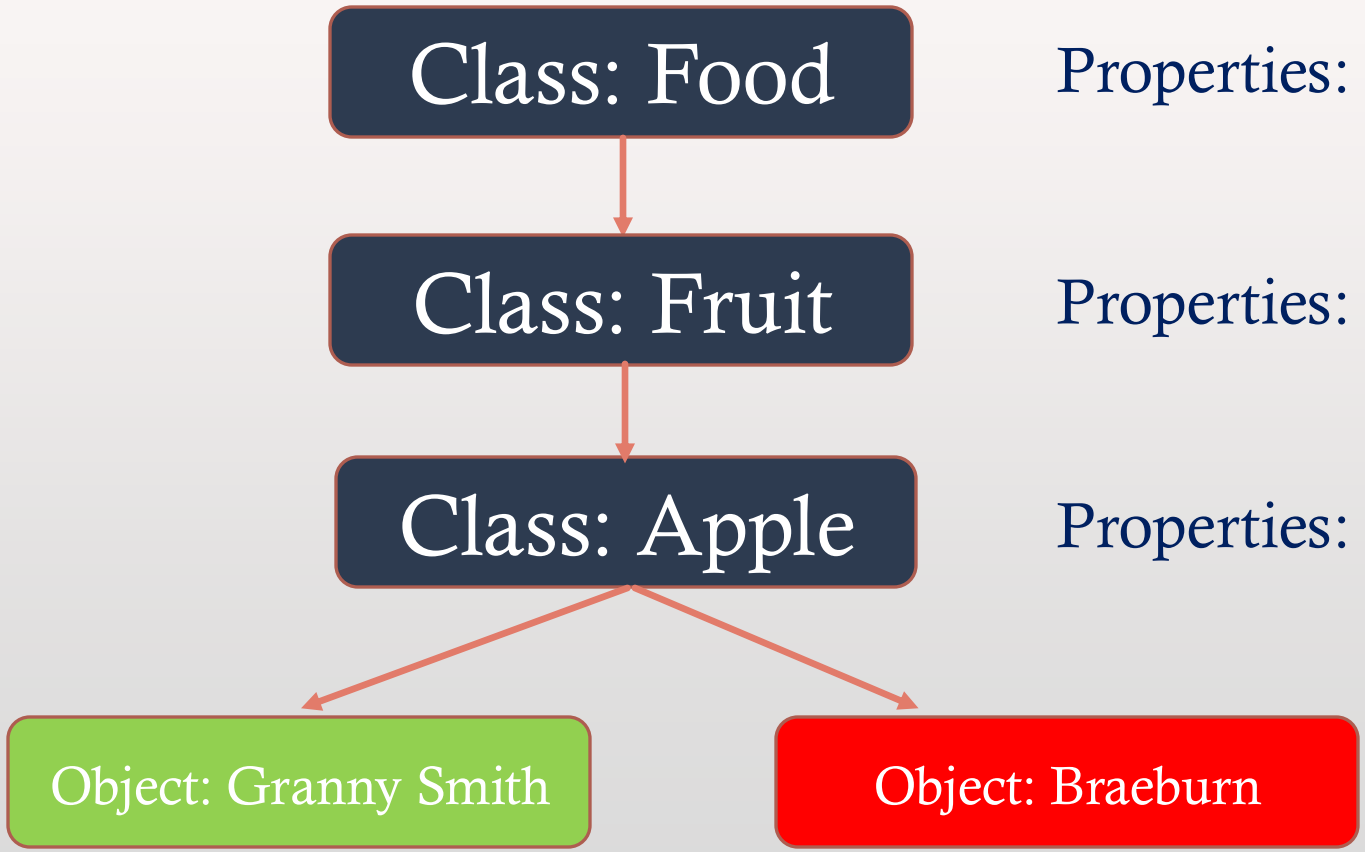
Encapsulation: Better control

- Programmes are made of code and data
- Encapsulation relates to the binding of a set of code and data within a single unit (class)
- This **object** can be private or public
- The complexity of the code and data is hidden from the user: **abstraction**

Inheritance: better re-use of code

- An object can acquire the properties of another object
- Consider a Braeburn apple object. It has properties {edible, nutritious, seeded, sweet, red}
- Now consider a Granny Smith apple object. It has properties {edible, nutritious, seeded, sweet, green}
- Inheritance allows us to avoid repetition in our work, and provides a way of grouping similar objects together

Inheritance



Properties: Edible, nutritious

Properties: Seeded, plant-based

Properties: Sweet, round

Polymorphism: Better flexibility

- Polymorphism means that objects of different classes can be treated as objects of a common base class. It allows you to write code that can work with objects of various types in a consistent way.

- Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks.

- Example: the Animal class has a ‘Noise’ function. An inheriting ‘Dog’ class can override this ‘Noise’ function to produce a ‘bark!’, while an inheriting sheep class can override the function to produce a ‘baa!’

- This concept allows our code to work with objects in a consistent way while maintaining flexibility

- A coding example is only needing one command to act on separate lists of integers, floats, and characters

C++ is flexible. You can use OOP, but you don’t have to

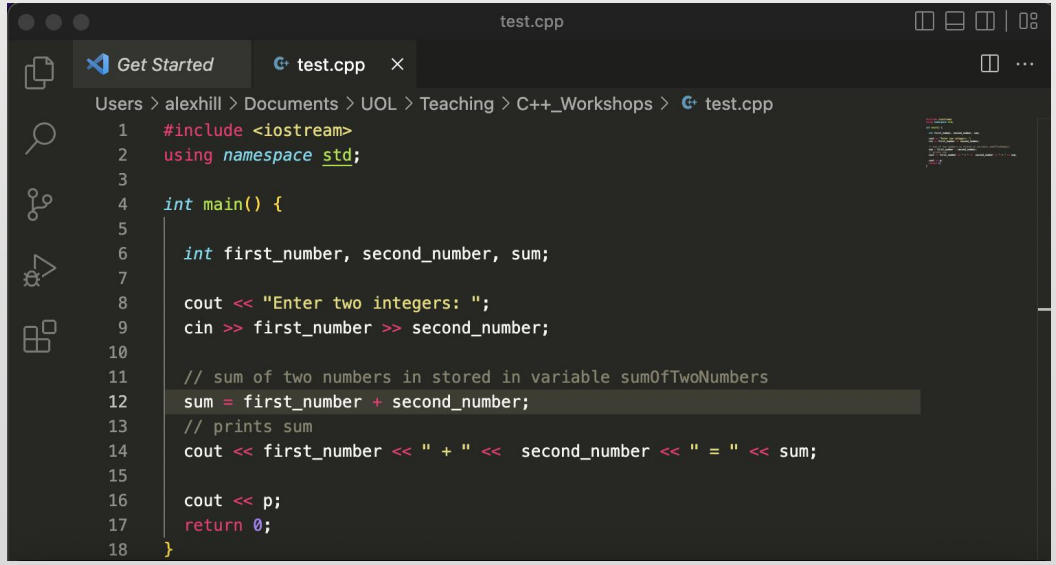
Getting started with C++

LIV.INNO

IDE – Integrated Development Environment

test.cpp

```cpp
#include <iostream>
using namespace std;

int main() {

  int first_number, second_number, sum;

  cout << "Enter two integers: ";
  cin >> first_number >> second_number;

  // sum of two numbers in stored in variable sumOfTwoNumbers
  sum = first_number + second_number;
  // prints sum
  cout << first_number << " + " <<  second_number << " = " << sum;

  cout << p;
  return 0;
}
```



- Provides an editor, compiler, debugger, AI co-pilot, package installer…
- I use an IDE (Visual Studio Code) as a fancy text editor, as it enables things like multi-line editing, autocomplete, useful colouring of different syntax

COMPILE

Compilers convert the **human-readable source code** that you write into something **readable by the CPU** (e.g. machine code or byte code)

```
#include <iostream>
int main() {
int a = 5;
int b = 10;
int sum = a + b;
std::cout << "The sum of a and b is: " << sum
<< std::endl;
return 0;
 }
```

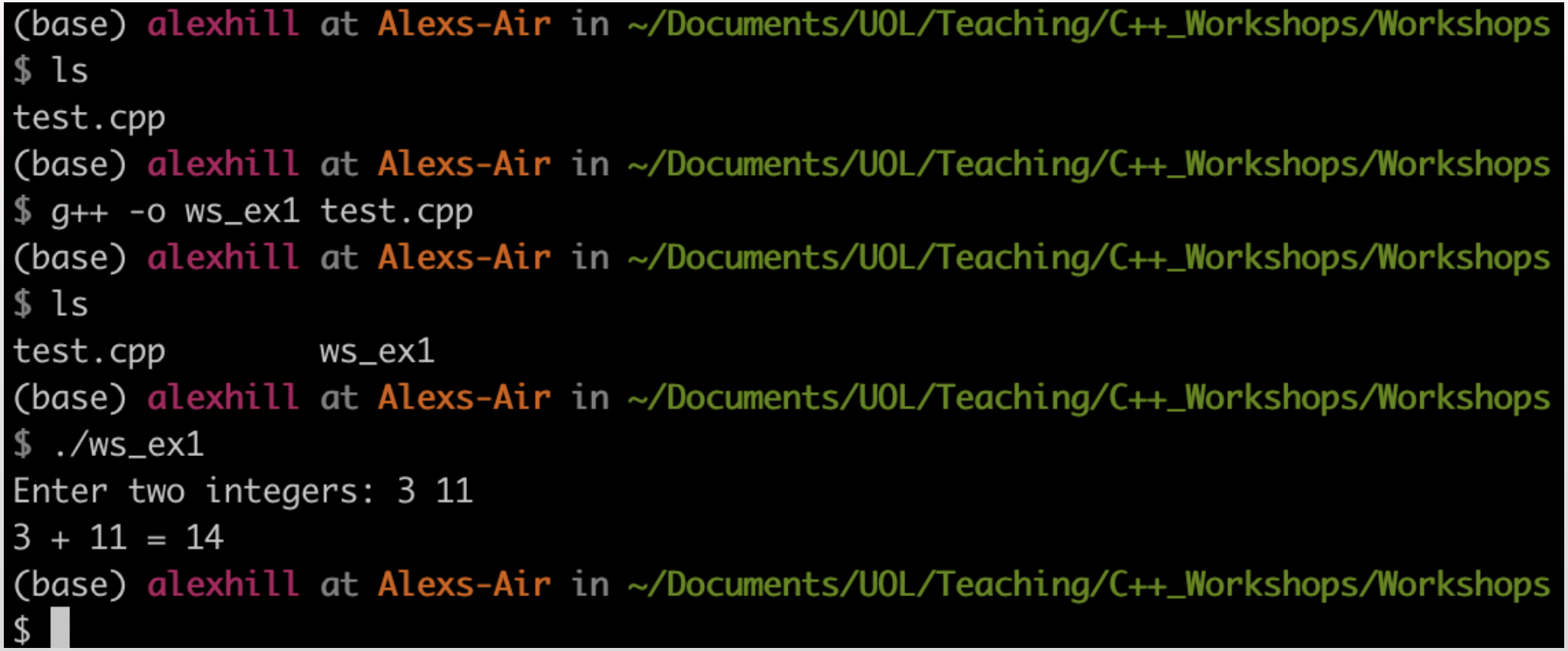
Compiler

```
.global main
main:
push rbp
mov rbp, rsp
mov DWORD PTR [rbp-20], 5
mov DWORD PTR [rbp-16], 10
mov eax, DWORD PTR [rbp-20]
add eax, DWORD PTR [rbp-16]
mov DWORD PTR [rbp-12], eax
lea rdi, [rip+15] # Address of the string
"The sum of a and b is: "
call std::operator<<(std::basic_ostream<char,
std::char_traits<char> >&, char const*)
mov rax, QWORD PTR [rip+26] # Address of std::cout
mov rsi, QWORD PTR [rip+38] # Address of the variable 'sum'
mov rdi, rax
call std::basic_ostream<char, std::char_traits<char> >::operator<<(int)
mov esi, 10
mov rdi, rax
call std::basic_ostream<char, std::char_traits<char> >::operator<<(int)
mov esi, 10
mov rdi, rax
call std::basic_ostream<char, std::char_traits<char> >::operator<<(int)
mov esi, 10
mov rdi, rax
call std::basic_ostream<char, std::char_traits<char> >::operator<<(int)
mov rdi, rax
call std::basic_ostream<char, std::char_traits<char> >::operator<<(char)
mov rdi, rax
call std::basic_ostream<char, std::char_traits<char>
>::operator<<(std::basic_ostream<char, std::char_traits<char>
>&(*)(std::basic_ostream<char, std::char_traits<char> >&))
mov eax, 0
pop rbp
ret
```

COMPILERS

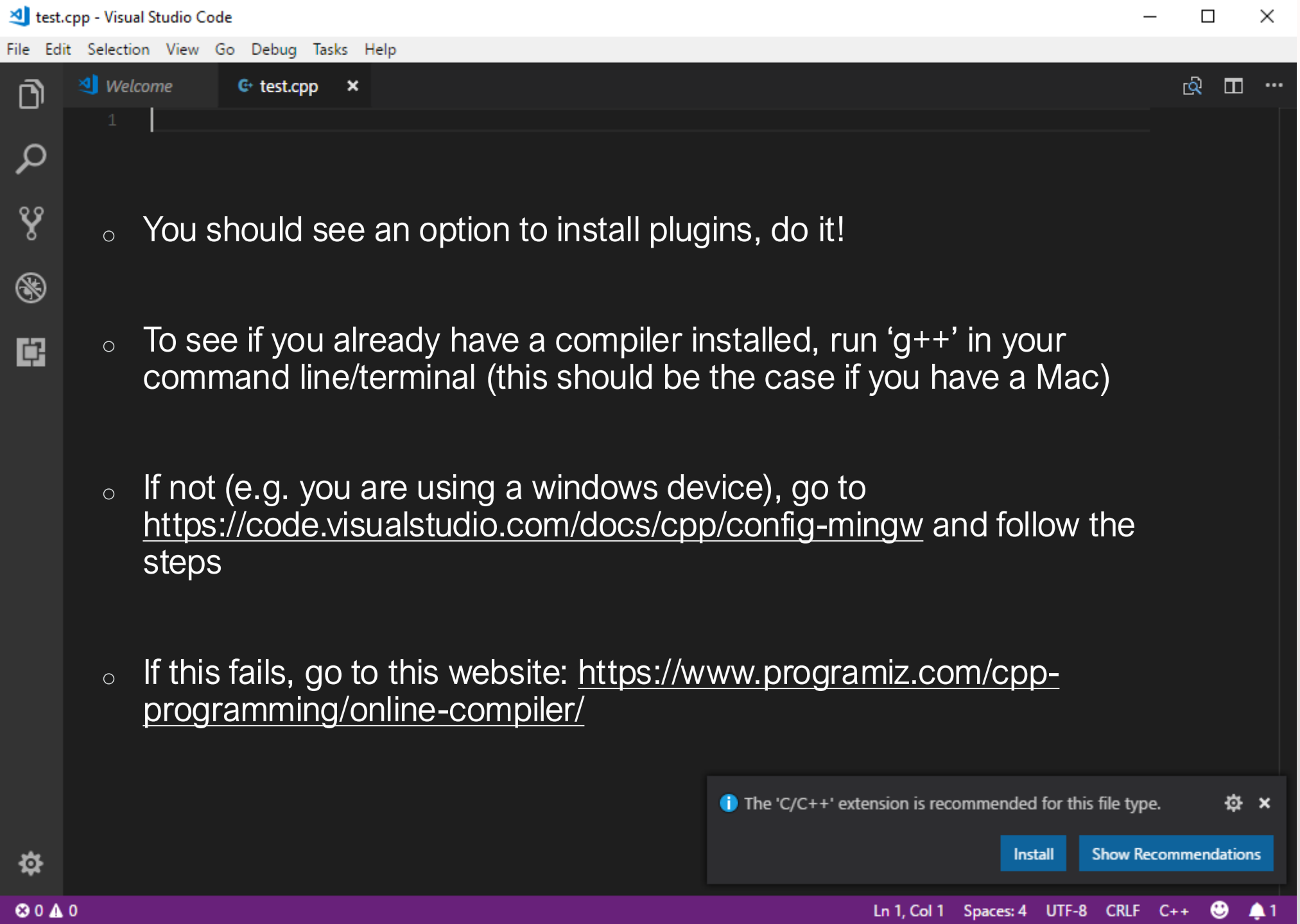
- I will use g++ (GNU C++)
- In order to run a C++ script, you first have to compile it
- $ g++ -o executable_name test.cpp
- $ ./executable_name

example


(base) alexhill at Alexs-Air in ~/Documents/UOL/Teaching/C++_Workshops/Workshops
$ ls
test.cpp
(base) alexhill at Alexs-Air in ~/Documents/UOL/Teaching/C++_Workshops/Workshops
$ g++ -o ws_ex1 test.cpp
(base) alexhill at Alexs-Air in ~/Documents/UOL/Teaching/C++_Workshops/Workshops
$ ls
test.cpp ws_ex1
(base) alexhill at Alexs-Air in ~/Documents/UOL/Teaching/C++_Workshops/Workshops
$ ./ws_ex1
Enter two integers: 3 11
3 + 11 = 14
(base) alexhill at Alexs-Air in ~/Documents/UOL/Teaching/C++_Workshops/Workshops
$

Getting started…

- Download an IDE, I suggest Visual Studio Code, which supports C++, C#, Java, Python, and others
  - https://code.visualstudio.com/download
- Create a new directory somewhere called Workshops, inside which create a file called ‘test.cpp’
- In Code: File > Open Workspace… > Open ‘Workshops’ > Open test.cpp

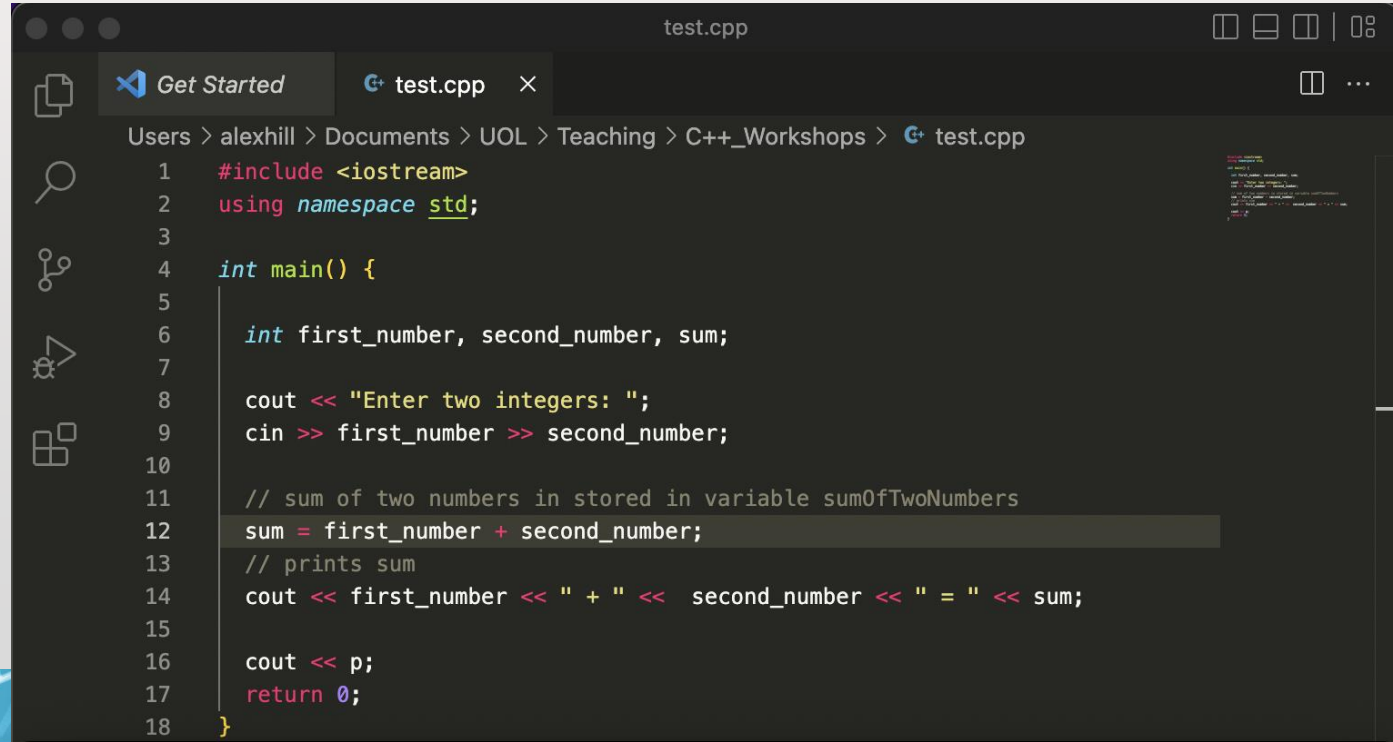


- You should see an option to install plugins, do it!
- To see if you already have a compiler installed, run ‘g++’ in your command line/terminal (this should be the case if you have a Mac)
- If not (e.g. you are using a windows device), go to https://code.visualstudio.com/docs/cpp/config-mingw and follow the steps
- If this fails, go to this website: https://www.programiz.com/cpp-programming/online-compiler/

Getting started…

- We'll start off with a simple code that creates an executable which takes two numbers and prints out the sum

- Or copy from https://alex-hill94.github.io/#WS1



Running the script

- Navigate to the Workshops folder in your terminal (or cmd line)
- Or https://www.programiz.com/cpp-programming/online-compiler/
- Try compile the script: **g++ -o test_it test.cpp**
- If using a Mac, run compiled script with: **./test_it**
- If using Windows, run compiled script with: **test_it.exe**

Congratulations/commiserations!

What's happening here?

```cpp
#include <iostream>
using namespace std;

int main() {

  int first_number, second_number, sum;

  cout << "Enter two integers: ";
  cin >> first_number >> second_number;

  // sum of two numbers in stored in variable sumOfTwoNumbers
  sum = first_number / second_number;
  // prints sum
  cout << first_number << " + " <<  second_number << " = " << sum << "\n";
  return 0;

}
```

What's happening here?

```
1    #include <iostream>
```

C++ allows the use of **headers**, files which store pre-defined functions

For this examples, the header <iostream> is needed to support the C++ I/O system

This header is included with your compiler, no further downloads are needed

Headers are included with the **#include** command

What's happening here?

```
2    using namespace std;
```

This tells the compiler to use the **std namespace**

A namespace is a way to encapsulate code elements (variables, functions, methods) into distinct packages

Elements are grouped into named containers, and are separate from other namespaces

This helps in larger projects, ensuring that there are no conflicts when using several libraries or collaborating on larger projects

The **std** namespace is the entire Standard C++ library

```
// Define a namespace called "MyNamespace"
namespace MyNamespace {
int myVariable = 42;
void myFunction() {
// Code for the function
}
}

int main() {
// Access elements within the namespace using the scope
resolution operator ::
int x = MyNamespace::myVariable;
MyNamespace::myFunction();
return 0;
}
```

```cpp
#include <iostream>
#include <cmath>
// using namespace std;

int main() {

int first_number, second_number, sum;
std::cout << "Enter two integers: ";
std::cin >> first_number >> second_number;

// sum of two numbers in stored in variable sumOfTwoNumbers
sum = first_number + second_number;
// prints sum
cout << first_number << " + " << second_number << " = " << sum << "\n";
return 0;

}
```

What's happening here?

```
4    int main() {
```

main() is the only function which must
be included in every C++ programme

This is where the programme execution begins

{ } indicates the start and end of the **main()** functions code

int specifies the type of data that **main()** will return (integer)

What's happening here?

```
6    int first_number, second_number, sum;
```

This defines three variables which will be called within the **main()** function.

Variables must be defined before use, i.e. given a designated data type

These are defined to have an *integer* data type

Notice that all C++ statements end with a semicolon

What's happening here?

```
8       cout << "Enter two integers: ";
```

This is a console output statement

It causes 'Enter two integers' to be printed on the screen

This is achieved with the output operator: **<<**

cout is a pre-defined identifier which stands for console output

"Enter two integers:" is a string, identified with double quotes

What's happening here?

```
9       cin >> first_number >> second_number;
```

This is a console input statement

Following the previous statement, the user is prompted to enter two integers into the terminal

This is achieved with the input operator: **>>**

cin is a pre-defined identifier which stands for console input

The previously defined variables **first_number** & **second_number** are assigned the values inputted

What's happening here?

```
11      // sum of two numbers in stored in variable sumOfTwoNumbers
```

This is a single line comment

`//` tells the compiler to ignore this sentence

These are used by developers to comment on code

In this case, it tells the reader what the function is doing

What's happening here?

```
12      sum = first_number + second_number;
```

This gives the variable **sum** the value of **first_number + second_number**

This is achieved with the addition operator **+**

What's happening here?

```
14    cout << first_number << " + " <<  second_number << " = " << sum << "\n";
```

This console output commands prints the stored variables and some strings

This results in a human-readable text output of the operation undertaken by the program

"**\n**" is a command that tells the console to go to the next line

What's happening here?

```
15      return 0;
16
17  }
```

This terminates **main()** and causes it to return a value of 0 to the calling process

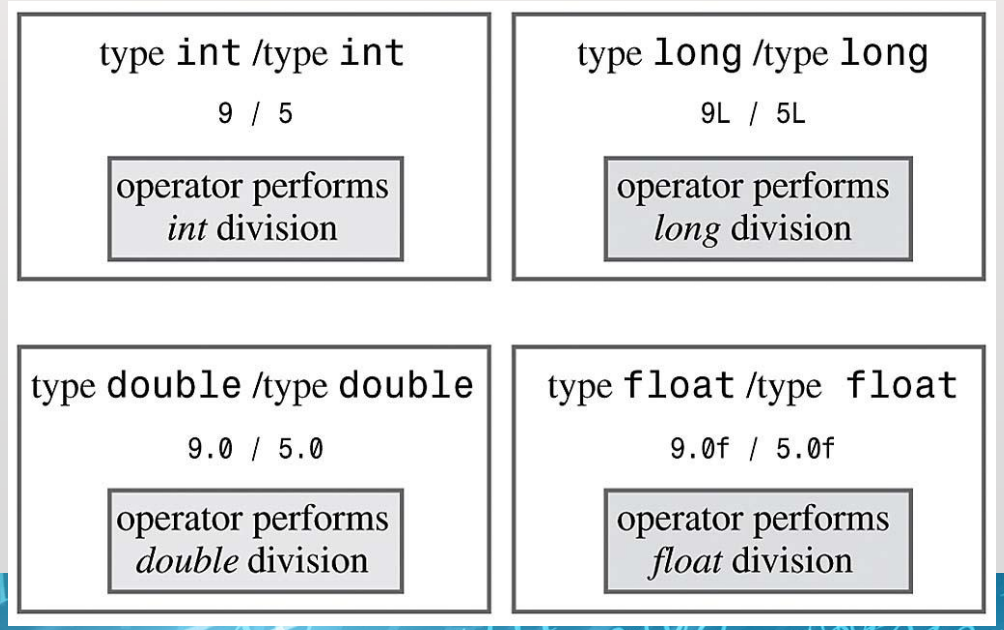
All your programs should return 0 when they terminate normally

} is the formal end of the program

Next steps

- Play around with this script using other operators (e.g. +, -, /, *) to get a feel for how they work

- Use different data types (e.g. int, float) and see how the operators interact with them



Next Week(s)

- Data Types
- Loops
- Functions
- Plotting Data

Homework

- Get your compiler working (if it isn't already) and run a basic script

Thanks!

LIV.INNO