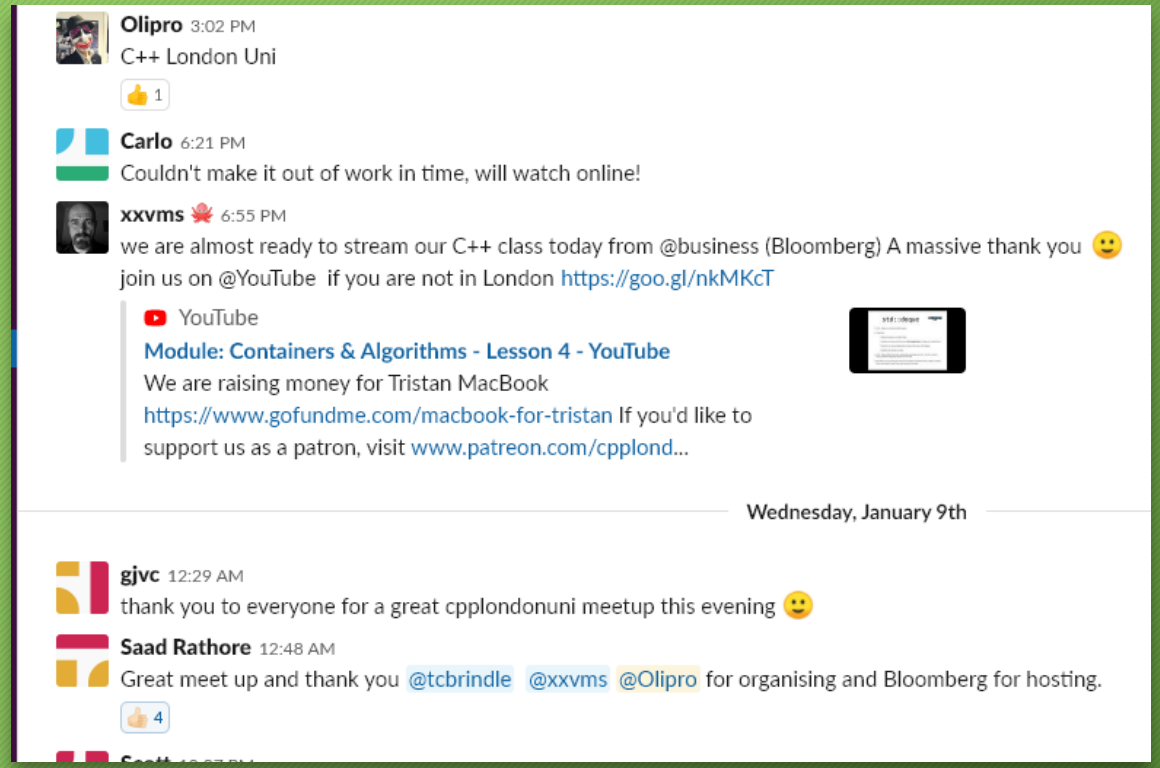
Initial C++ — Session 3

Tristan Brindle

Feedback

- We'd love to hear from you!
- The easiest way is via the *CPPLang* Slack organisation. Our chatroom is #cpplondonuni
- If you already use Slack, don't worry, it supports multiple workgroups!
- Go to https://slack.cpp.al to register.



Last week

- Functions revision
- Introduction to using variables
- Passing variables to functions
- If statements

This week

- Getting set up for C++
- If statements revision
- A brief introduction to std::vector
- Loops in C++

Getting set up for C++

- So far we have only used an online compiler for our examples and exercises
- In order to write real programs, we need to have a C++ compiler installed our our own systems
- Today we'll get everyone set up with a free compiler (either GCC or Clang)
- We'll also install an *integrated development environment* (IDE) called CLion, which we can use to write our programs
- If you need any help **please ask**!

Step 1: installing a compiler

- Windows:
  - Go to https://nuwen.net/mingw.html and download “mingw-16.1.exe”
  - Run the downloaded executable, and select a location to install — the default C:\MinGW works well
- Mac OS:
  - Open a Terminal window and type

    ```
    xcode-select --install
    ```
  - *(note two dashes before install!)*
  - Follow the prompts that appear

Step 2: installing CLion

- Go to https://www.jetbrains.com/clion/ and download CLion via the “start a 30 day trial” link
- Run the installer and then launch CLion
- The wizard should guide you through the process of creating a “toolchain”
  - Windows: if prompted, tell it that you want to use MinGW, and enter the directory you selected earlier (e.g. C:\MinGW)
  - Mac and Linux: CLion should automatically detect your compiler

Step 3: make sure everything is working

- In CLion, go to `File->New Project`
- In the window that appears, make sure “C++ executable” is selected on the left and then click “Create”
- Wait for CLion to finish setting up your project (this may take a while the first time)
- Click the “Play” button on the toolbar (or select Run->Run from the menus)
- This will build and run your project. If you see a message saying “Hello world” then congratulations, you’re all set up!


C++ London
Uni

Revision: if statements

- One of the basic building blocks of programs is the *if statement*
- This tests some condition, and performs some some instructions *if* the condition is true
- The basic form of an if statement in C++ is

```
if (condition) {
    // do something
}
```

Revision: if statements

- We can also add else if to test a second condition:

```
if (condition) {
    // do something
} else if (other condition) {
    // do something else
}
```

- We can have as many else if statements as we like
- Conditions are tested in the order that they appear

Revision: if statements

- Finally, we can add an else statement as a fallback if none of the other conditions are true:

```
if (condition) {
    // do something
} else if (other condition) {
    // do something else
} else {
    // do a third thing
}
```

If statement example

```
void greet(std::string name) {
    if (name == “Tom") {
        std::cout << "Hello Tom";
    } else if (name == "Oli") {
        std::cout << "Hello Oli";
    } else {
        std::cout << "I don't know you";
    }
}

int main() {
    greet(“Tom");

    greet("Steven");

    return 0;
}
```

Last week's exercise: FizzBuzz

- In Wandbox, write a function named `fizzbuzz` which takes an `int` as input
  - If the int is exactly divisible by 3, print "fizz"
  - If the int is exactly divisible by 5, print "buzz"
  - If the int is exactly divisible by *both* 3 and 5, print "fizzbuzz"
  - Otherwise, print "Not fizzy or buzzy"
- Try calling this function from `main()` with different values, e.g. 99, 125, 225, 1024…

My solution

```
void fizzbuzz(int i) {
    if (i % 15 == 0) {
        std::cout << "fizzbuzz\n";
    } else if (i % 3 == 0) {
        std::cout << "fizz\n";
    } else if (i % 5 == 0) {
        std::cout << "buzz\n";
    } else {
        std::cout << "not fizzy or buzzy\n";
    }
}

int main() {
    fizzbuzz(99);  fizzbuzz(125);
    fizzbuzz(225); fizzbuzz(1024);

    return 0;
}
```

A very brief introduction to std::vector

- One of the most useful types provided by the standard library is `std::vector`
- A `std::vector` is a *collection of values* of some other type
- For example, we can have a vector of `int`s, or a vector of `std::strings`
  - Vector is an example of a *generic type*
- To use `std::vector`, we need to say `#include <vector>` at the top of our source code

Using std::vector

- When we use a vector, we need to specify what type it will hold
- We do this by writing the type in angle brackets `< >`
- For example `std::vector<int>` is a vector of `int`s
- We can add new *elements* to a vector using the `push_back()` command
- We can get the number of elements in the vector using the `size()` command

```
#include <iostream>
#include <string>
#include <vector>

int main()
{
    std::vector<int> integers = {1, 2, 3};

    std::vector<std::string> strings = {
        "one", "two", "three"
    };

    strings.push_back("four");

    std::cout << strings.size() << '\n';
}
```

Using std::vector

- We can access elements of the vector by their *index* by using square brackets [ ]
- **Important**: the first element is at index 0!
- **Super important**: if you try to access an index which is out of range (i.e. the vector does not have that many elements), ***bad things will happen***
- (It may crash, or it may continue "working" but with bogus data)

```
int main()
{
    std::vector<std::string> strings = {
        "one", "two", "three"
    };

    std::cout << strings[0] << '\n';
    // prints "one"!

    strings[2] = "forty four";
}
```

Using std::vector

- We can perform an action for each element of a vector by using the `for` keyword
- This is an example of a “range-for loop”, which we’ll talk more about shortly
- In this case, the variable `i` holds a *copy* of the element we are looking at

```
int main()
{
    std::vector<int> my_vec = {1, 2, 3};

    for (int i : my_vec) {
        std::cout << i << ' ';
    }
    // prints 1 2 3
}
```

Exercise

- Create a new C++ executable project in CLion
- In `main()`, create a vector of strings called `names`, and initialise it with the names of the people on your table
  - Don't forget to `#include <vector>`!
- Use the `push_back()` command to add the names "Tom", "Oli" and "Michael" to your `names` vector
- Print out the number of elements in names
- Print out each name in turn, separated by a newline

My solution

```
#include <iostream>
#include <vector>

int main()
{
    std::vector<std::string> names = {"Arthur", "Beatrice", "Clive"};
    names.push_back("Tom");
    names.push_back("Oli");
    names.push_back("Michael");

    std::cout << "I have " << names.size() << " names\n";

    for (std::string name : names) {
        std::cout << name << '\n';
    }
}
```

Homework

- Write a program that reads in a sequence of ten `floats` from the user using `std::cin`. Print out the *minimum* and *maximum* values that they entered. Can you do this without storing every entered value?
- Extend your program so that it also prints out the *mean* of the numbers the user entered
  - Hint: this time you may want to use a std::vector to store the input values to make the calculation easier
- Extend your program so that it also prints out the *median* of the numbers the user entered
- (Harder): Extend the program so that it prints out the *mode* (that is, the value that appears most often) of the input sequence.
  - Hint: there may be more than one such value

Thank You!

As usual, we will be going to the pub! Support us @ https://patreon.com/CPPLondonUni