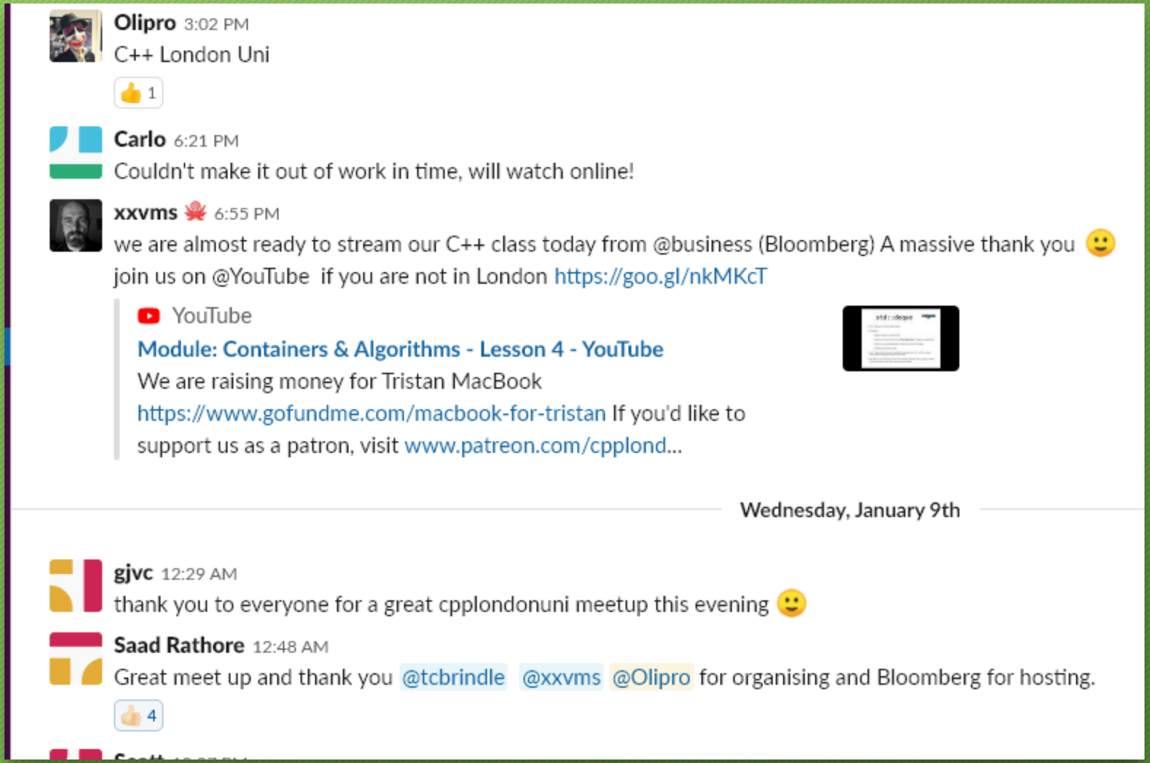
Initial C++ — Session 4

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.



Olipro 3:02 PM
C++ London Uni
👍 1

Carlo 6:21 PM
Couldn't make it out of work in time, will watch online!

xxvms 6:55 PM
we are almost ready to stream our C++ class today from @business (Bloomberg) A massive thank you 🙂 join us on @YouTube if you are not in London https://goo.gl/nkMKcT

YouTube
Module: Containers & Algorithms - Lesson 4 - YouTube
We are raising money for Tristan MacBook https://www.gofundme.com/macbook-for-tristan If you'd like to support us as a patron, visit www.patreon.com/cpplond...

Wednesday, January 9th

gjvc 12:29 AM
thank you to everyone for a great cpplondonuni meetup this evening 🙂

Saad Rathore 12:48 AM
Great meet up and thank you @tcbrindle @xxvms @Olipro for organising and Bloomberg for hosting.
👍 4

Last week

- Installing a compiler and IDE
- If statements revision
- A brief introduction to std::vector

This week

- Loops in C++
    - While, for and range-for
    - Break and continue

- Passing arguments by reference

Last week's homework

- Write a program that reads in a sequence of ten `float`s 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

My solution (min and max)

```cpp
#include <iostream>

// Get min and max of 10 floats
int main() {
    float value = 0;
    std::cin >> value;

    float min = value;
    float max = value;

    for (int i = 0; i < 9; ++i) {
        std::cin >> value;
        if (value < min) {
            min = value;
        }
        if (value > max) {
            max = value;
        }
    }

    std::cout << "Min was " << min << '\n';
    std::cout << "Max was " << max << '\n';
}
```

My solution (min and max improved)

```cpp
int main()
{
    float value = 0.0f;
    if (!(std::cin >> value)) { return -1; }

    float min = value;
    float max = value;

    for (int i = 0; i < 9; ++i) {
        if (!(std::cin >> value)) { return -1; }
        min = value < min ? value : min;
        max = value > max ? value : max;
    }

    std::cout << "Min was " << min << '\n';
    std::cout << "Max was " << max << '\n';
}
```

My solution (mean)

```cpp
int main()
{
    float sum = 0.0f;

    for (int i = 0; i < 10; ++i) {
        float value = 0.0f;
        if (!(std::cin >> value)) { return -1; }
        sum += value;
    }

    std::cout << "Mean was " << sum/10 << '\n';
}
```

My solution (median)

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

int main()
{
    std::vector<float> values;

    for (int i = 0; i < 10; ++i) {
        float value = 0.0f;
        if (!(std::cin >> value)) { return -1; }
        values.push_back(value);
    }

    std::sort(values.begin(), values.end());

    std::cout << "Median was: " << (values[4] + values[5])/2.0f << '\n';
}
```

My solution (mode)

```
#include <algorithm>
#include <iostream>
#include <map>

int main()
{
    std::map<float, int> counts;

    for (int i = 0; i < 10; ++i) {
        float value = 0.0f;
        if (!(std::cin >> value)) {
            return -1;
        }

        counts[value] += 1;
    }
```

```
    // First, find out the maximum count
    // int max_count = counts.crbegin()->second;
    int max_count = 0;
    for (std::pair<float, int> key_value : counts)
    {
        max_count =
          std::max(max_count, key_value.second);
    }

    // Now print out all keys with that count
    std::cout << "Modal values: ";
    for (std::pair<float, int> key_value : counts)
    {
        if (key_value.second == max_count) {
            std::cout << key_value.first << ' ';
        }
    }
    std::cout << '\n';
}
```

Loops

- Another of the basic building blocks of programs are *loops*
- A loop is an instruction that says "do something repeatedly"
- C++ has two main types loops: `while` loops and `for` loops
- We've already seen one use of `for` with vectors

While loops

- The basic form of a while loop is

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

- This first tests whether the condition is true: if so, it executes the instructions in the *body* of the loop
- It then tests the condition a second time: if it is still true, it executes the instructions again and so on, *while* the condition is true
- Q: What is the minimum number of times this loop may run?
- A: Zero (If the condition is false the first time we test it)

While loop example

```cpp
#include <iostream>

int main()
{
    int i = 0;
    while (i < 5) {
        std::cout << i << ' ';
        ++i; // shorthand for i = i + 1
    }
}
// prints 0 1 2 3 4
```

Do-while loops

- A while loop may also be written

```
do {
    // something
} while (condition);
```

- This time the condition is tested *after* the loop body has been executed
- Q: What is the minimum number of times this loop will run?
- A: *One*

While versus do-while example

```cpp
int main() {
    int i = 1;
    while (i < 1) {
        std::cout << "i = " << i << ' ';
        ++i;
    }

    int j = 1;
    do {
        std::cout << "j = " << j << ' ';
        ++j;
    } while (j < 1);
}
// prints j = 1
```

For loops

- Recall our first while loop example:

```
int i = 0;           // (1)
while (i < 5) {      // (2)
    /* ...do stuff... */
    ++i;             // (3)
}
```

- This kind of loop is so common that C++ has a shorthand for it: the `for` statement
- A `for` statement has three parts: the *initialiser (1)*, *condition (2)*, and *increment (3)*
- For example, we can write the above as

```
for (int i = 0; i < 5; ++i) {
    /* ...do stuff... */
}
```

For loops

- For loops are most often used with integers (as in the previous example), but this is not required
- In particular, any of the *initialiser, condition* or *increment* may be empty
    - An empty condition is always `true`
- Q: What does this do?

```
for ( ; ; ) { ... }
```

- A: Runs forever! (Equivalent to `while (true) { … }`)
- Remember a for loop is *always* just shorthand for a while loop!

For loop example

```cpp
int main()
{
    int sum = 0;
    for (int i = 0; i < 5; ++i) {
        sum = sum + i; // or sum += i;
    }
    std::cout << "Sum is " << sum << '\n'l
}
// prints “Sum is 10”
```

Range-for loops

- Finally, C++ also has *range-for loops*
- These are used to execute some code *for each element* in some *collection*
  - For example, each element in a `std::vector`
  - Other languages often call this *foreach* or something similar
- In order to be usable in a range-for loop, the collection type must meet the requirements of the C++ *range "protocol"*
- All the standard library containers meet this requirement

Range-for loops

- We write a range-for loop as

```
for (type element : range) {
    /* do stuff */
}
```

- Note the use of a colon (:) not a semicolon (;)!
- You can think of this as roughly equivalent to

```
for (int i = 0; i < range.size(); ++i) {
    type element = range[i];
    /* do stuff */
}
```

- So a range-for loop is shorthand for a for loop, which is shorthand for a while loop!

Range-for loop example

```
int main()
{
    std::vector<float> vec = { 1.234f, 3.142f, 2.172f, -987.654f };
    float product = 1.0f;

    for (float val : vec) {
        product *= val;
    }

    std::cout << "Product is " << product << '\n';
}
```

Exercise

- Create a new project in CLion
- In your main() function, create a vector of strings containing with the names of each person on your table
- Use a *range-for loop* to print out every element of the vector
- Modify your program to instead use a "non-range" *for loop* to print every element of the vector
- Modify your program to use a *while loop* instead of a for loop

My solution

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

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

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

    for (int i = 0; i < names.size(); ++i) {
        std::cout << names[i] << '\n';
    }

    int i = 0;
    while (i < names.size()) {
        std::cout << names[i] << '\n';
        ++i;
    }
}
```

Controlling loop iteration

- Sometimes we need greater control over exactly how our loops operate
- For example, we may want to skip some elements, or stop processing a loop “early”
- C++ provides two keywords for this purpose: `break` and `continue`
- `break` is used to *immediately exit from a loop*
- continue is used to *immediately skip to the next iteration*
- These can be used with any loop construct (while, for, range-for)

Break statement example

```cpp
#include <iostream>
#include <string>

int find_space(std::string str) {
    int pos = 0;
    for (char c : str) {
        if (c == ' ') {
            break;
        }
        ++pos;
    }
    return pos;
}

int main() {
    std::string hello = "Hello World";
    std::cout << "Space in position " << find_space(hello) << '\n';
}
// prints "Space in position 5"
```

Continue statement example

```cpp
int count_words(std::vector<std::string> words) {
    int num_words = 0;
    for (std::string word : words) {
        if (word.empty()) { // equivalent to word.size() == 0
            continue;
        }
        ++num_words;
    }
    return num_words;
}

int main() {
    std::vector<std::string> words = {
        "", "aardvark", "", "", "banana", "coffee", "", ""
    };
    std::cout << "Vector contains  " << count_words(words) << " words\n";
}
// prints "Vector contains 3 words"
```

Exercise

- Write a program which reads in a sequence of strings from the user with `std::cin`, and stores them in a `std::vector`
- Continue reading strings until the user enters the string "quit"
- Once the user has typed "quit", print out all the strings that they entered, EXCEPT those which begin with the letter 'b'.
    - For example, given the input

      apricot banana cherry date quit

      your program should print

      apricot cherry date

Thank You!

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