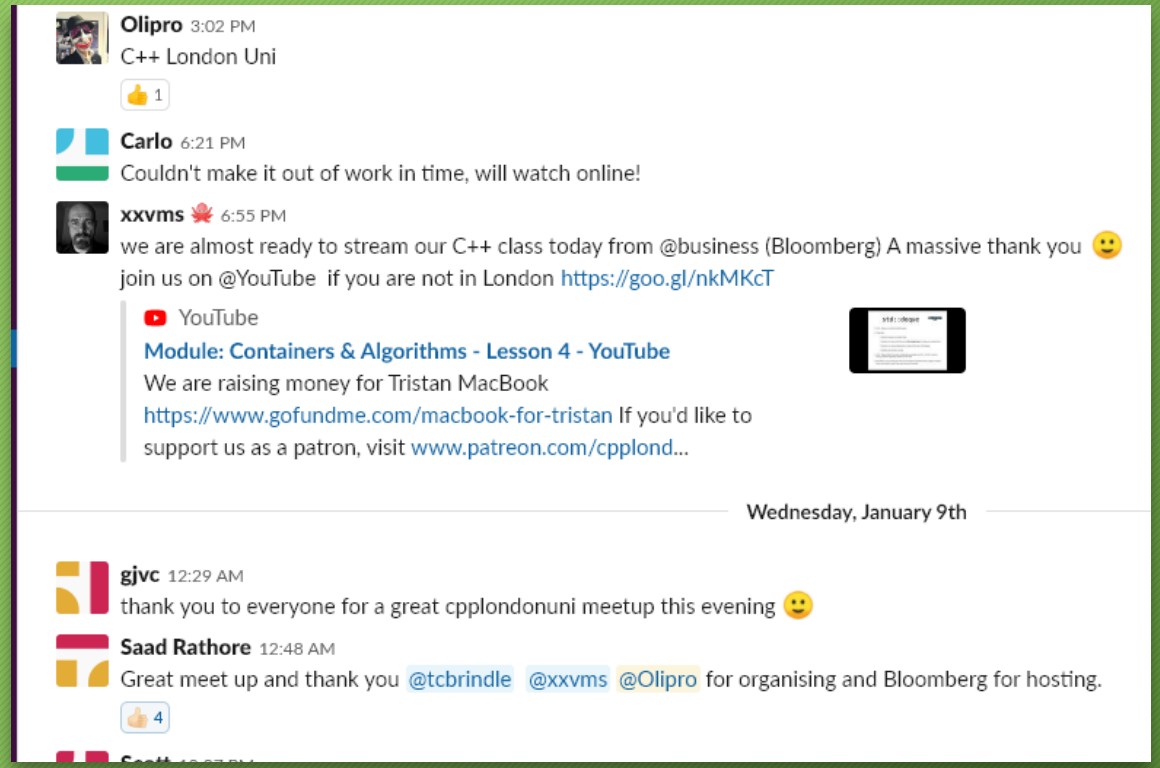
Custom Types — Session 1

Tristan Brindle

Copyright 2019 - CPP London Uni C.I.C.

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.



Copyright 2019 - CPP London Uni C.I.C.

Last week

- Revision: const and references
- Pass-by-reference-to-const
- Const, references and auto
- Namespaces

Copyright 2019 - CPP London Uni C.I.C.

This week

- End-of-module C++ *questionnaire*

- Introduction to custom types
  - Defining simple structs
  - Creating variables of struct type
  - Passing structs to functions

Copyright 2019 - CPP London Uni C.I.C.

Exercise

- Write a function `print_string()` that takes a read-only (const) reference to a `std::string` as an argument, and prints the string using `std::cout`
- Call your function with a local string variable. Verify that it works correctly.
- Call your function with a string literal. What happens? Why?
- Modify your function so that it takes the string argument by value instead. Does the code still compile? What happens? Why?
- Modify your function so that it takes the string argument by non-const reference instead. Does the code still compile? What happens? Why?

Copyright 2019 - CPP London Uni C.I.C.

Solution

```
void print_string1(const std::string& str)
{
    std::cout << str << '\n';
}

void print_string2(std::string str)
{
    std::cout << str << '\n';
}

void print_string3(std::string& str)
{
    std::cout << str << '\n';
}
```

```
int main()
{
    std::string hello = "Hello world";

    print_string1(hello);
    print_string1("Hi C++ London Uni");

    print_string2(hello); // okay (copies)
    print_string2("Hi C++ London Uni");

    print_string3(hello);
    print_string3("Hi C++ London Uni"); // ERROR!
}
```

Copyright 2019 - CPP London Uni C.I.C.

Homework/revision

1. Write a function lower_case() which takes a reference to a std::string, and modifies the string so that every character is changed to lower case.
    - For example, the string “MiXeD” should be changed to “mixed”.
2. Write a function headline_case() which takes a reference to a string. Change the first character of every word of the string into a capital letter.
    - For example, the string “this is a headline” should be changed to “This Is A Headline”
3. Write a function sentence_case() which changes the first character of every sentence into a capital letter. Sentences are separated by a full stop followed by a single space character.
    - For example, the string “this is sentence 1. this is sentence 2.” should be changed to “This is sentence 1. This is sentence 2.”
4. Write a function sentence_case2() which does the same as sentence_case(), except that sentences may be separated by an arbitrary number of whitespace characters.
    - For example, the string “sentence 1. \n\t\r sentence 2.” should be changed to “Sentence 1. \n\t\r Sentence 2.”
    - (HINT: the function std::isspace() can be used to check for whitespace characters.)
5. Write a function sentence_case3() which does the same as sentence_case2(), except that all whitespace characters between sentences are replaced with a single space character.
    - For example, the string “sentence 1. \n\t\r sentence 2.” should be changed to “Sentence 1. Sentence 2.”

Copyright 2019 - CPP London Uni C.I.C.

Solution

- See https://github.com/CPPLondonUni/initial_week5_homework_soln

Copyright 2019 - CPP London Uni C.I.C.

Quiz time!

- Yes, it's the end-of-module *definitely-not-a-test* happy quiz fun time!
- Please go to https://bit.ly/34cGVkh
- You'll have 30 minutes to complete the quiz
- Scores will be emailed to you privately over the next few days

Copyright 2019 - CPP London Uni C.I.C.

Quiz answers

- Over to Oli

Copyright 2019 - CPP London Uni C.I.C.

Revision: types

- In programming languages, a *type* is a way of giving meaning to some data
- The type of some data tells us *what it represents* and *what we can do with it*
- For example, we can multiply two numbers, but we cannot meaningfully multiply two strings

Copyright 2019 - CPP London Uni C.I.C.

Revision: types

- C++ has many built-in (“fundamental”) types such as int, float, double, bool etc
- The standard library has many other commonly-used types such as std::string and std::vector
- The language provides us with many tools which we can use to define our own data types
  - …which is exactly what the library uses to build things like std::string

Copyright 2019 - CPP London Uni C.I.C.

Data structures

- A data structure is (abstractly) a way to organise the data used by your program
- The choice of data structure can heavily affect the performance and usability of your program
- Selecting the right data structures and the defining the relationships between them is an essential programming skill

Copyright 2019 - CPP London Uni C.I.C.

> “I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.”
>
> — Linus Torvalds

Data structures

C++ London Uni

Custom types

- C++ provides us with several tools which we can use to define our own types, and implement new data structures, for example:
  - Structs/classes
  - Pointers
  - Enumerations
  - Arrays
  - Unions

Copyright 2019 - CPP London Uni C.I.C.

Custom types

- C++ provides us with several tools which we can use to define our own types, and implement new data structures, for example:
  - **Structs/classes**
  - Pointers
  - Enumerations
  - Arrays
  - Unions

Copyright 2019 - CPP London Uni C.I.C.

Structs

- A struct (or class) in C++ is a collection of *data members* (or *member variables*), along with *member functions* which operate on them
- We can define new struct types using the `struct` keyword
- Unlike some other languages, the keywords `struct` and `class` mean almost exactly the same thing in C++
  - The only difference is in which defaults you get
  - Today (and for the next few sessions) we'll be using the `struct` keyword, but I'll often use the two terms interchangeably

Copyright 2019 - CPP London Uni C.I.C.

Our first struct

- We can define a new struct type using the `struct` keyword
- Inside the struct definition we list its *data members*, similarly to how we define local variables in a function:

```
struct Point {
    int x = 0;
    int y = 0;
};
```

- A struct definition must always end with a semicolon!

Copyright 2019 - CPP London Uni C.I.C.

Our first struct

- A struct definition always introduces a new type, *distinct from any other type in our program*
- This means that two structs are *different types*, even if they have the same members:

```
struct First {
    int i = 0;
};
```

```
struct Second {
    int i = 0;
};
```

- Here, `First` and `Second` are *different types*

Copyright 2019 - CPP London Uni C.I.C.

Our first struct

- C++ places no restrictions on the *number* or *types* of struct member variables
- However, member variables of reference type (e.g. `int&`) can have surprising effects and are best avoided
  - Consider using pointers (which we'll cover later) or `std::reference_wrapper` instead
- Similarly, member variables of const type (e.g. `const std::string`) don't mix well with move semantics and are also best avoided
  - We'll talk much more about move semantics later in the course

Copyright 2019 - CPP London Uni C.I.C.

Our first struct

- Once we have defined our new type, we can create *variables* or *objects* of that type
  - We sometimes call this an *instance* of a particular type
- For simple structs, we initialise the *data members* of an instance using curly braces:

```
struct Point {
    int x = 0;
    int y = 0;
};

Point p  = { 1, 2 };
```

- (Note that this only applies to "simple" (*aggregate*) types — structs and classes with constructors or private data work differently. Initialisation in C++ is very complicated 😟)

Copyright 2019 - CPP London Uni C.I.C.

Our first struct

- We can access the members of a struct instance using a `.` (dot) after the variable name, for example

```
Point p = {1, 2};
p.x = 5; // p is now {5, 2}
```

- We can use a struct type anywhere that we can use a fundamental type, for example:
  - as a member variable of another struct type
  - as a function parameter (by reference or by value)
  - as the element type of a `std::vector`

Copyright 2019 - CPP London Uni C.I.C.

Struct example

```
struct Example {
    std::string str = "default";
    int value = -1;
};

int main()
{
    Example ex1 = { "a", 1 };
    Example ex2{"b", 2}; // may omit the '='
    Example ex3{};    // e.str == "default", e.value = -1

    const Example ex4{"d", 4}; // may declare an Example as const
    ex4.str = "Hello"; // ERROR: e4 is const
    ex3.str = "Hello"; // okay, e3 is not const

    Example& ex_ref = ex1;    // may declare a reference-to-Example
    ex_ref.value = 22;       // may access members via a reference
    std::cout << ex1.value << '\n'; // what does this print?
}
```

Copyright 2019 - CPP London Uni C.I.C.

Passing structs to functions

- As with fundamental types, we may use struct types as function parameters, and return struct instances from functions
- Reminder: C++ uses *pass-by-value* by default!
  - Struct instances are *copied* by when passed by value
- If we wish to pass by reference, we need to explicitly say so using the Type& syntax
- We can of course also form read-only (const) references to struct instances too, just like with fundamental types

Copyright 2019 - CPP London Uni C.I.C.

Struct example (2)

```cpp
struct Example {
    int value = 0;
};

void set_val(Example ex, int new_val) {
    ex.value = new_val;
}

void set_val2(Example& ex, int new_val) {
    ex.value = new_val;
}

int main() {
    auto ex = Example{3}; // may use auto, if we are explicit in the initialiser

    set_val(ex, 99);
    std::cout << ex.value << '\n';
    // prints 3

    set_val2(ex, 99);
    std::cout << ex.value << '\n';
    // prints 99
}
```

Copyright 2019 - CPP London Uni C.I.C.

Exercise

- In the main.cpp file of a new CLion project, define a new struct called `Student`
- A Student should have two member variables, both of type `std::string`, named `first_name` and `surname`
- Write a function `void print_surname(const Student& s)` which prints the surname of the given student
- Check that your function works correctly
- Extension: create a `std::vector` of `Students`. Use a range-for loop to print the surname of each student

Copyright 2019 - CPP London Uni C.I.C.

Solution

```
struct Student {
    std::string first_name = "";
    std::string surname = "";
};

void print_surname(const Student& s) {
    std::cout << s.surname << '\n';
}

int main() {
    const Student tom{"Tom", "Breza"};
    print_surname(tom);

    std::vector<Student> students{
            tom,
            {"Oli", "Ddin"},
            Student{“Tristan", "Brindle"}
    };

    for (const auto& s : students) {
        print_surname(s);
    }
}
```

Copyright 2019 - CPP London Uni C.I.C.

Thank You!

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

Copyright 2019 - CPP London Uni C.I.C.