C++ London University Session 1

Tristan Brindle

Welcome to C++ London University!

C++ London University:

- Website: cpplondonuni.com
- Github: github.com/CPPLondonUni

Where to find Tom Breza:

- On Slack: cpplang.slack.com #learn #cpplondon
- E-mail: tom@PCServiceGroup.co.uk
- Mobile: 07947451167

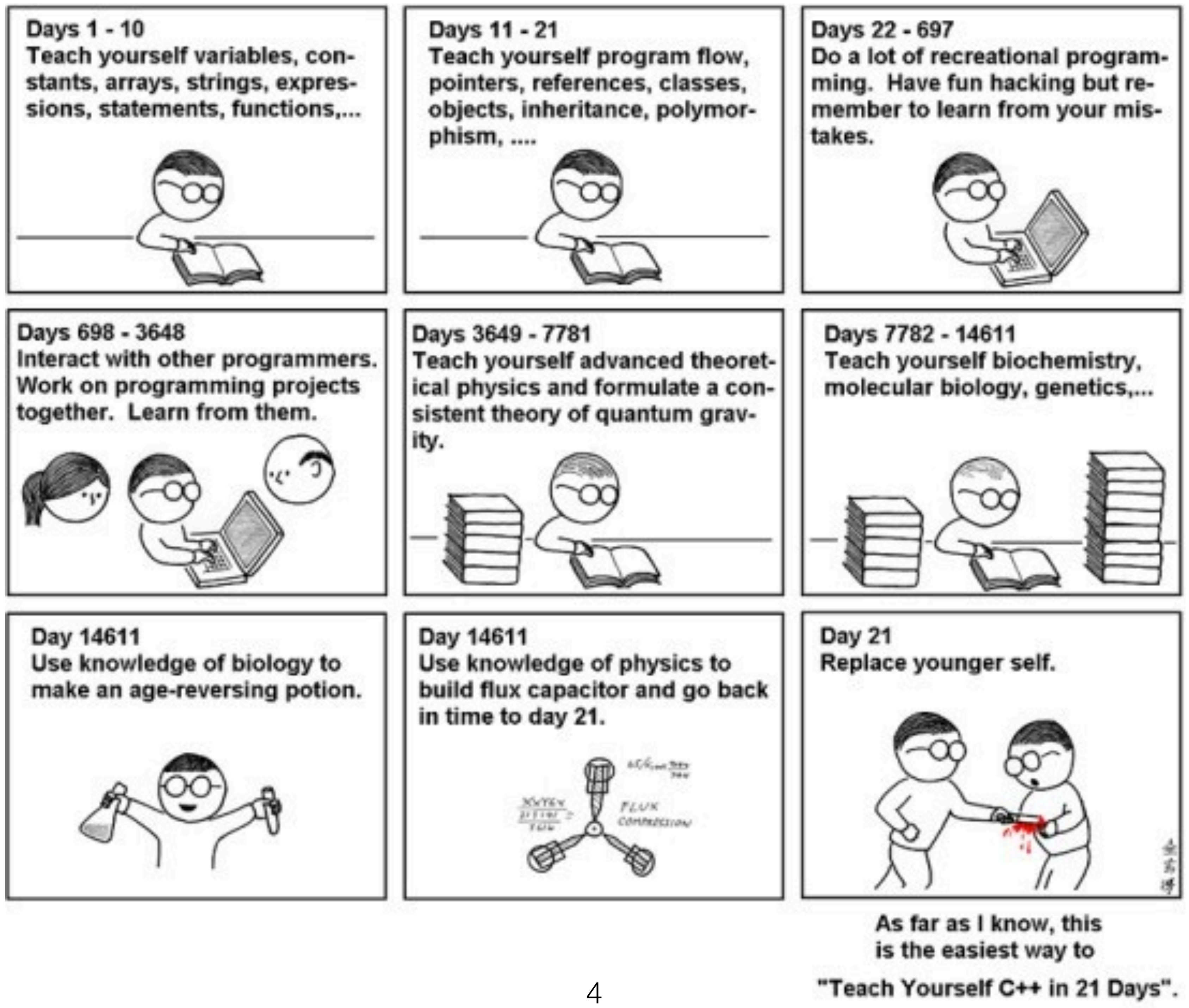
My stuff:

- Website: tristanbrindle.com
- Twitter: @tristanbrindle
- Github: github.com/tcbrindle

About these sessions

- An introduction to C++
- We can’t turn you into an expert in 5 weeks (sorry!)
- …but we’ll try to give you enough information to get started

“Teach yourself C++ in 21 days”


Days 1 - 10
Teach yourself variables, constants, arrays, strings, expressions, statements, functions,...
Days 11 - 21
Teach yourself program flow, pointers, references, classes, objects, inheritance, polymorphism, ...
Days 22 - 697
Do a lot of recreational programming. Have fun hacking but remember to learn from your mistakes.
Days 698 - 3648
Interact with other programmers. Work on programming projects together. Learn from them.
Days 3649 - 7781
Teach yourself advanced theoretical physics and formulate a consistent theory of quantum gravity.
Days 7782 - 14611
Teach yourself biochemistry, molecular biology, genetics,...
Day 14611
Use knowledge of biology to make an age-reversing potion.
Day 14611
Use knowledge of physics to build flux capacitor and go back in time to day 21.
FLUX COMPRESSION
Day 21
Replace younger self.
As far as I know, this is the easiest way to "Teach Yourself C++ in 21 Days".

About these sessions

- An introduction to C++
- We can’t turn you into an expert in 5 weeks (sorry!)
- …but we’ll try to give you enough information to get started
- We might try for a beginner/intermediate split depending on feedback

(Preliminary) Lesson Plan

- Week 1 (today!): "Hello World" - introducing the main() function, writing output, strings and functions and variables.
- Week 2: More about types, and classes, methods and operator overloading.
- Week 3: Pointers, references, inheritance, polymorphism
- Week 4: Basic templates, containers, smart pointers
- Week 5: More about the standard library and algorithms and putting it all together

(Preliminary) Lesson Plan

- Week 1 (today!): "Hello World" - introducing the main() function, writing output, strings and functions and variables.
- Week 2: More about types, and classes, methods and operator overloading.
- Week 3: Pointers, references, inheritance, polymorphism
- Week 4: Basic templates, containers, smart pointers
- Week 5: More about the standard library and algorithms and putting it all together

A word about feedback

- Your feedback is vital
- Otherwise, we don’t know what you don’t know!
- If you don’t know, please **ASK**

Why C++?

- Usually because it’s *fast*
  - Direct access to hardware
  - Zero-overhead abstractions
  - Efficient resource usage
- Used everywhere
  - Everything from micro controllers to supercomputers
  - Games, financial trading, web browsers, etc etc etc

Why not C++?

- Usually because it’s *hard*
    - Partly true unfortunately
    - C++ allows access to low-level facilities
    - C++ has lots of features — use them wisely
    - Some warts and “gotchas” due to its age
- …but it’s not *that* hard! 🙂

A (very) brief history of C++

- 1979: Bjarne Stroustrup starts work on “C with Classes”
- 1983: C with Classes renamed C++
- 1990: ISO committee formed to standardise C++
- 1998: First standard version released (C++98)
- 2011: Major update to the standard (C++11)
- 2014, 2017: Further standard updates (C++14, C++17)
- 2020, 2023….?

“Modern” C++

- C++11 changed the game
- Don’t bother learning C++98
- Make sure any textbooks or online resources you use are teaching you *today’s* C++.

Any questions before we move on?

Hello World

Hello World

- Exercise 1

1. Go to wandbox.org
2. Enter the following

```cpp
// Our first C++ program!

#include <iostream>

int main()
{
    std::cout << "Hello world\n";

    return 0;
}
```

3. Click “Run”

Deconstructing Hello World

Deconstructing Hello World

```
// Our first C++ program!
```

- This is a comment
- Inline comments start with two slashes (`//`) and continue to the end of the line
- Multiline comments start with `/*` and end with `*/`

Deconstructing Hello World

```
#include <iostream>
```

- This line tells the compiler to include the contents of "`iostream`" in our program
- "`iostream`" is provided by the standard library and contains code to let us write to (and read from) the console
- `#include` is used to break large programs into smaller, manageable pieces, and to use code from other libraries (as we've done here)

Deconstructing Hello World

```
int main()
```

- This line declares a function called "`main`" which returns an `int`(-eger) and takes no parameters
- Every C++ executable contains a `main` function, which is where the program starts.
- `main()` has some special rules

Deconstructing Hello World

```
{
```

- A curly brace opens a *block*
- In this case, the block contains the *definition* of our `main()` function
- Blocks control object lifetimes in C++, as we'll see later

Deconstructing Hello World

```
std::cout << "Hello world\n";
```

Deconstructing Hello World

```
std::cout << "Hello world\n";
```

- `cout` (“console output”) is an object provided by the standard library for printing text
- As part of the standard library, it belongs to the `std` *namespace*, so we write `std::` to access it
- Later we’ll see a shortcut to avoid having to type `std::` everywhere, but use it with caution.

Deconstructing Hello World

```
std::cout << "Hello world\n";
```

- The << symbol means (in this case) “pass the thing on the right to the *output stream* on the left”
- This is an example of *operator overloading* in C++
- Later, we’ll see other meanings of <<, and how to define the meaning of operators for our own types

Deconstructing Hello World

```
std::cout << "Hello world\n";
```

- This is a string literal
- The `\n` at the end means “start a new line here”
- Sometimes you’ll see `(std::)endl` used as an alternative way to start a new line

Deconstructing Hello World

```
std::cout << "Hello world\n";
```

- Every C++ statement ends with a semicolon
- If you forget it, the compiler will usually tell you…
- …but if you get strange errors, check that you've got your semicolons right

Deconstructing Hello World

```
        return 0;
```

- The `return` keyword tells the program to leave the current function, returning the value (in this case `0`) to the caller
- By convention, returning zero from `main()` tells the operating system that the program ran successfully, any other value indicates an error
- Remember how I said `main()` was special….?

Deconstructing Hello World

```
}
```

- This closes the block we opened earlier
- When we leave a block, local variables defined in that block get destroyed
- This is the single best thing about C++ (really!)

Any questions before we move on?

Functions

- C++ programs are composed of *functions*, small pieces of reusable code
- We've already seen the `main()` function
- The small print: In C++, functions come in two kinds, *member functions* and *non-member functions* ("*free functions*"). Today we're talking about non-member functions; we'll discuss member functions ("methods") next time.

Functions (2)

- The general form of a function declaration is

  ```
  return-type function-name(param-type param-name, …)
  ```

- Every function in C++ returns zero or one value(s)

- If the function does not return a value, then the return type is `void`

Functions (3)

- For example, we can define a function which adds two `int`s like so:

```
int add(int a, int b)
{
    return a + b;
}
```

- This defines a function “add” which takes two parameters named *a* and `b` (both of type `int`) and returns a value of type `int`

Functions (4)

- To *call* (run) a function, we say `function_name(arguments)`, e.g

```
std::cout << add(3, 4) << '\n'; // prints 7
```

- In C++ a function must be *declared* before it can be called

Exercise 2

- Exercise 2
  - In your “hello world” program, write a function

    ```
    void hello_cpp_london_uni()
    ```

    which prints “Hello C++ London University” to the console
  - Call this function from your `main()`

Solution 2

```
void hello_cpp_london_uni()
{
    std::cout << "Hello C++ London University\n";
}

int main()
{
    hello_cpp_london_uni();
}
```

Any questions before we move on?

Variables

- Dictionary definition: (roughly) “a named storage location for some data”
- In C++, every variable has a *type*, which dictates what sort of data it can hold
- The data currently held in a variable is called its *value*
- In C++, the *lifetime* of a variable is usually tied to the scope (block) in which it is declared

Declaring Variables

- To declare a variable, we can say

```
type-name variable-name = initialiser;
```

- e.g.

```
int i = 0;
```

- (There are a couple of other initialisation forms we'll see later when we discuss classes)

- **Always** initialise your variables

Declaring Variables (2)

- C++11 added *type deduction*, so we could also say

```
auto variable-name = initialiser;
```

- e.g.

```
auto i = 0;
```

- Now the type of `i` is determined by its initialiser (still `int` in this case).
- This can be really handy, but (as ever) use with caution

Constants

- We can declare a variable to be a constant using the keyword `const` in front of the type name, for example

```
const int i = 0;
```

- When declared like this, the value of `i` cannot be changed after it is initialised
- This is helps reduce programming errors and (sometimes) allows better optimisation
- Pro tip: make variables "const by default", mutable only when necessary

Value Semantics

- Unlike many other programming languages, C++ uses *value semantics* rather than *reference semantics* by default
- This means (roughly) that copies of variables are distinct; changing the value of a copy will not affect the original variable (i.e. copies are “deep”).
- Later we’ll see how we can use references in C++

Example of lifetimes and value semantics

```
int i = 0;    // Create a variable i of type int with value 0
{
    int j = i; // Create another int j,
               //   initialised with a copy of i's value
    j = 3;     // j now has value 3, i is still 0
}              // j is destroyed here

// i = j;      // error, j does not exist any more
```

Any questions before we move on?

Strings

- So far the only type we've seen is `int`. This is a *fundamental type* (one built in to the language) representing a mathematical integer.
- The C++ standard library also provides us with many other useful types, such as `std::string`
- To use `std::string` we need to say `#include <string>` near the top of our source file

Strings (2)

- We can create a `std::string` in exactly the same way as we created an `int`

```
std::string hello = "Hello";
std::string world = " world";
```

- `std::string` has all sorts of useful functionality, for example we can concatenate (join) two strings by saying

```
const auto hello_world = hello + world;
```

- (This is another example of operator overloading)
- `std::string`s can be printed just like `int`s

```
std::cout << hello_world << '\n';
```

Exercise 3

- Exercise 3
  - Write a function `say_hello()` which takes a `std::string` parameter called name, and returns a string containing that name with “Hello ” in front
  - Use this function to print “Hello <your name>” from your `main()` routine, e.g. “Hello Tristan”

Solution 3

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

std::string say_hello(std::string name)
{
    const std::string hello = "Hello ";
    return hello + name;
}

int main()
{
    std::cout << say_hello("Tristan") << '\n';
}
```

Any questions before we wrap up?

Summary

- This was a only very brief introduction to the wonderful world of C++
- We've learned how "hello world" works
- We've learned about the `main()` function
- We've learned how to `#include` standard library headers
- We've learned how to write functions and declare variables
- We've been introduced to type deduction (`auto`) and `const`
- We've been introduced to the ideas of variable lifetime and value semantics
- We've been introduced to `std::string`

Next time

- Week 1: "Hello World" - introducing the main() function, writing output, strings and functions and variables.
- Week 2: More about types and classes, methods and operator overloading.
- Week 3: Pointers, references, inheritance, polymorphism
- Week 4: Basic templates, containers, smart pointers
- Week 5: More about the standard library and algorithms and putting it all together

“Homework”

- (Requires further reading)

  1. Read about `std::vector`. Modify your solution to Exercise 3 to print “Hello Tom”, “Hello Phil”, “Hello Tristan” on separate lines using a vector of strings and a range-for loop.

  2. Write a program to ask the user to enter their name at the console. Read this into a `std::string` using `std::cin`. If the name is one of “Tom”, “Phil”, “Tristan” or your name then print “Hello <name>!” (e.g. “Hello Tom!”), otherwise print “Hello stranger!”

Online Resources

- https://isocpp.org/get-started
- cppreference.com — The bible, but aimed at experts
- cplusplus.com — Another reference site, also has a tutorial section
- learncpp.com — Free online tutorial, very up-to-date
- https://www.pluralsight.com/authors/kate-gregory - Comprehensive set of courses from an experienced C++ trainer (free trial)
- reddit.com/r/cpp_questions
- Cpplang Slack channel — https://cpplang.now.sh/ for an “invite”
- StackOverflow (but…)

Thanks for coming!

C++ London University:

- Website: cpplondonuni.com
- Github: github.com/CPPLondonUni

Where to find Tom Breza:

- On Slack: cpplang.slack.com #learn #cpplondon
- E-mail: tom@PCServiceGroup.co.uk
- Mobile: 07947451167

My stuff:

- Website: tristanbrindle.com
- Twitter: @tristanbrindle
- Github: github.com/tcbrindle

See you next time! 🙂