C++ London University

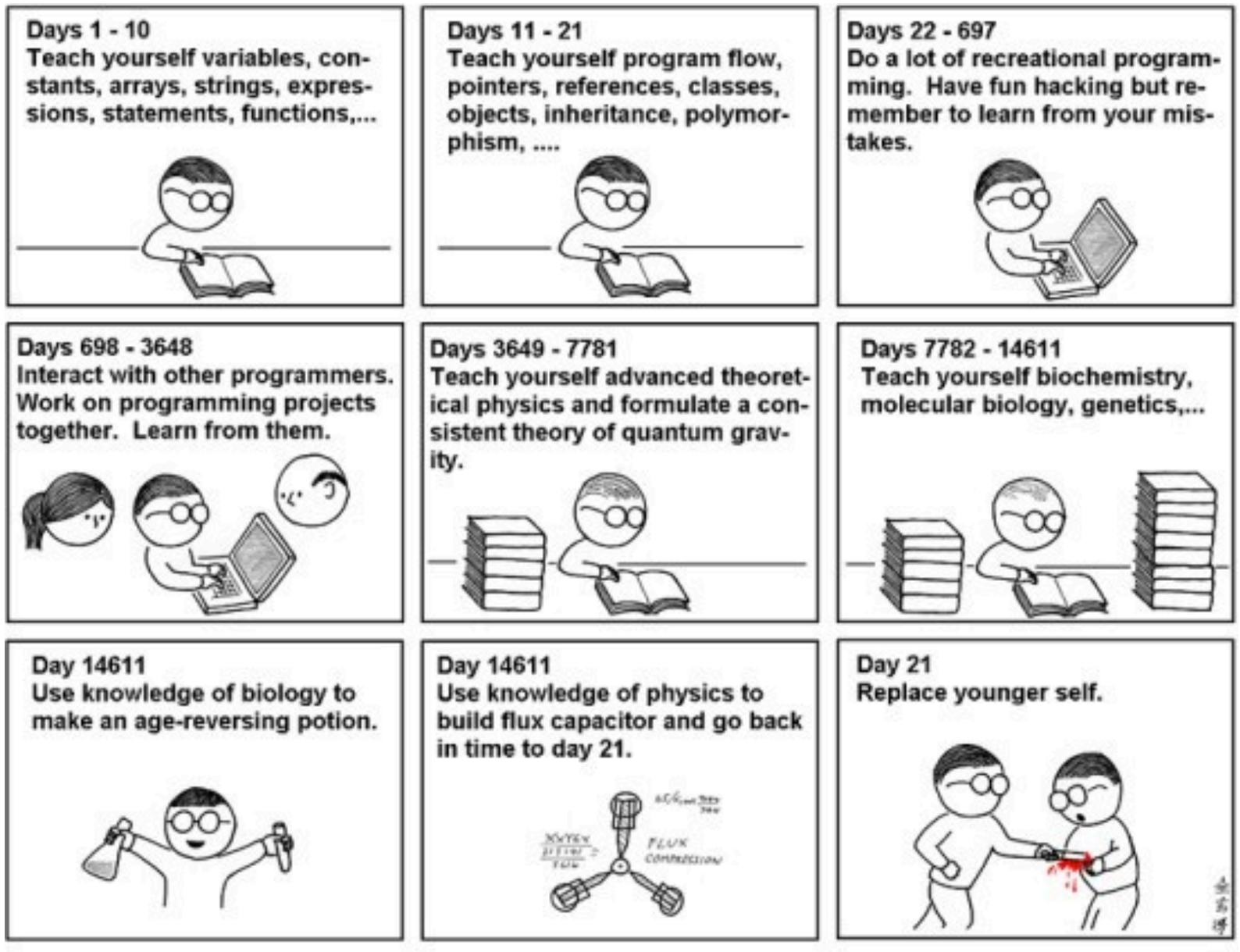
Session 1

Tristan Brindle

About these sessions

- An introduction to C++
- A mixture of talks, class exercises and homework
- We can’t turn you into an expert (sorry!)
- …but we’ll try to give you enough information to get started

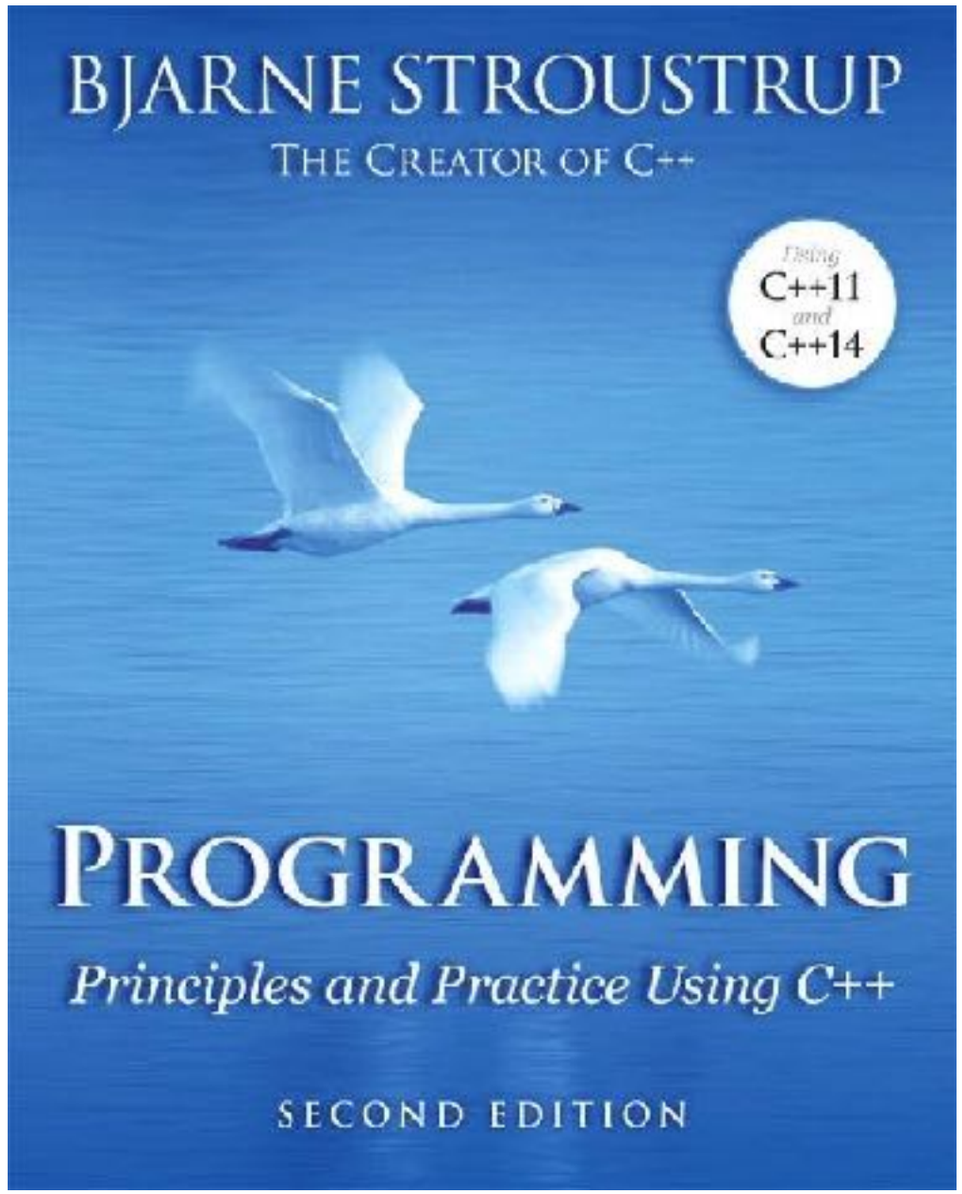
“Teach yourself C++ in 21 days”



Textbook

- We'll be using *"Programming Principles and Practice Using C++"* by Bjarne Stroustrup

- Please pick up a copy



Feedback

- We’d love to hear from you!
- The easiest way is via the *cpplang* channel on Slack — we have our own chatroom, *#ug_uk_cpplondonuni*
- Go to https://cpplang.now.sh/ for an “invitation”

Today's lesson plan

- Introduction to C++
- Deconstructing "Hello World"
- Types
- Functions in C++

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?

Exercise 1

1. Go to wandbox.org
2. Enter this text
3. Click “Run”

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

#include <iostream>

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

    return 0;
}
```

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 the `iostream` *header* 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 correct

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?

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

Types

- C++ has many built-in (“fundamental”) types, such as `int`, `float`, `double`, `bool` etc

- The standard library has lots more commonly-used types such as `std::string` and `std::vector`

- The language provides us with many tools to define our own types, which we’ll learn about as the course progresses

Types

- C++ is a *statically-typed* programming language
- This means that every variable has its type determined when the program is compiled
- C++'s *type safety* means that we can catch many potential errors before the program is run
- There are ways around the type safety rules, but avoid them if at all possible

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

- Note: C++ has two kinds of functions, *member functions* and *non-member functions*. Today we're taking about non-member ("free") functions. We'll talk about member functions ("methods") later in the course.

Functions

- The usual 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

- 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

- 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

- In your “hello world” program in Wandbox, write a function

  ```
  void hello_cpp_london_uni()
  ```

  which prints “Hello C++ London Uni” to the console

- Call this function from your `main()`

Solution 2


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

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

Exercise 3

- In Wandbox, write a function `say_hello()` which takes a parameter of type `std::string` called `name`, and returns a string containing that name with “Hello ” in front

- Use this function to print “Hello <your name>” from `main()`, e.g. “Hello Tristan”

- You will need to add `#include <string>` near the top of your program to use `std::string`

Solution 3


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

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

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

Any questions before we wrap up?

Summary

- This was only a very brief introduction to the wonderful world of C++
- We've learnt how "hello world" works
- We've learnt about types and type-safety
- We'v learnt how to `#include` standard library headers
- We've learnt how to define functions
- We've been introduced to `std::string`

Next time

- Intro to C++ part 2
- More on types, `const` and `auto`
- Variable declarations
- Value semantics and RAII
- Control flow

Homework

- Get a copy of “Programming Principles and Practice using C++”

- Read chapters 2 and 3

- If you’re confident to do so, install CLion IDE (jetbrains.com/clion)

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 Uni:

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

Where to find Tom Breza:

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

My stuff:

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

See you next time! 🙂