A very brief introduction to C++

Mario Alvarez-Picallo
mario.alvarez-picallo@cs.ox.ac.uk

Department of Computer Science, University of Oxford

May 1, 2017

Some context

What to expect:

- A first introduction to C++
- You will know enough to write not-so-simple programs
- If you want to learn more, you will know where to look

What *not* to expect

- A course on low-level programming
- A complete course on C++
- A course on AI/game development/web development/...

Above all, this is a course for beginners, no previous experience is required.

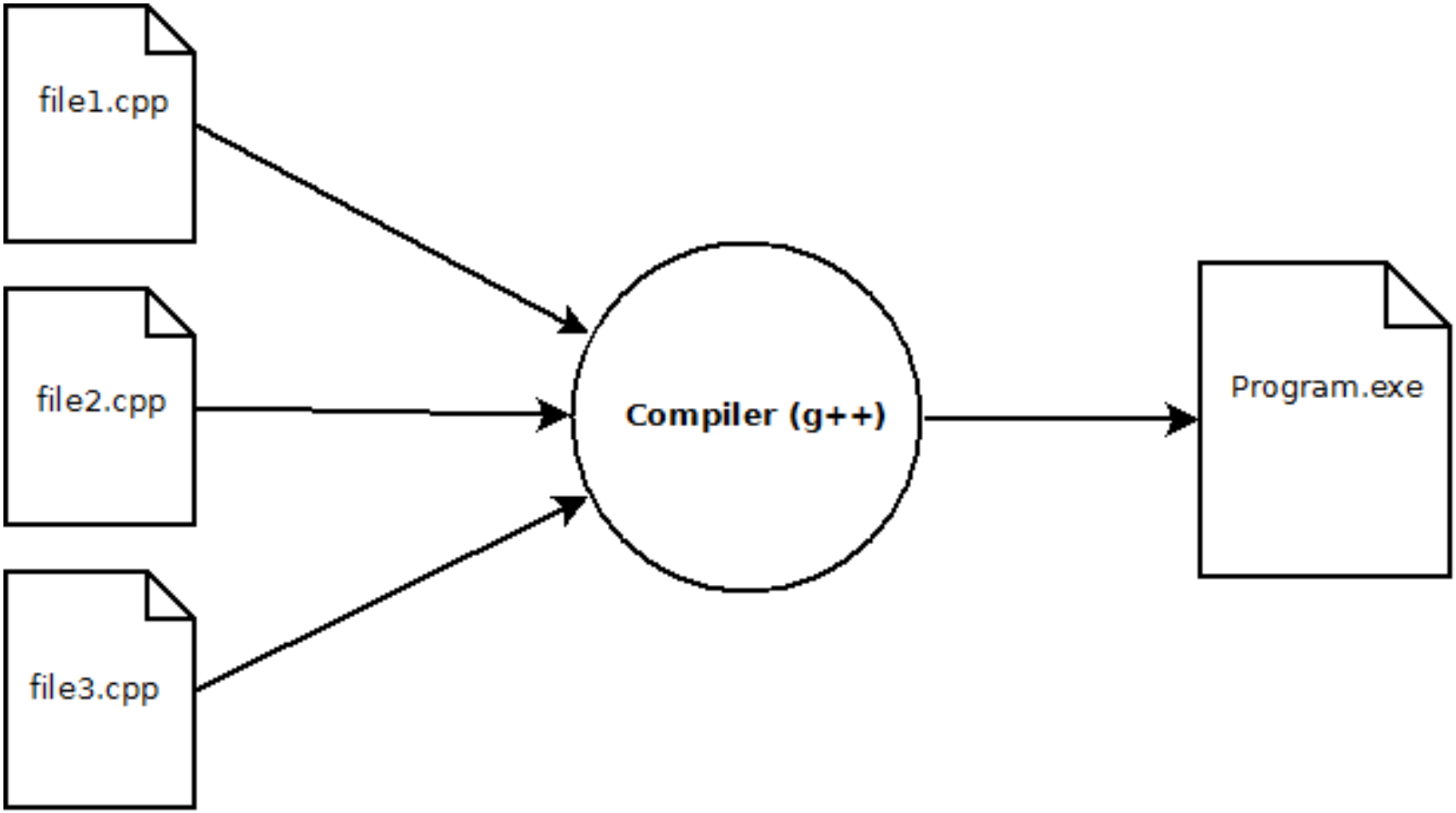
What is C++?

C++ is superficially similar to modern languages like JavaScript or Python - but it is a very different beast!

- A *low-level* language
  - None of the niceties of Python or JavaScript
  - A lot more *control*
  - C++ won't do anything for you
- A *compiled* language
  - There is no interpreter
  - Code needs to be *compiled* into a program before being run
  - This makes for *faster* code
- A *statically typed* language
  - More on this later!
- A *very complex* language
  - Lots of corner cases, lots of redundancy...
  - We won't deal with most of it!
  - For reference, use: http://cplusplus.com/reference

What is a compiler?

Long (*very* long) story short:



A first C++ program

```
#include <iostream>

int main()
{
  // This is a comment!
  std::cout << "Hello, World!";
  std::cout << std::endl;
  return 0;
}
```

A first C++ program

```
#include <iostream>

int main()
{
  // This is a comment!
  std::cout << "Hello, World!";
  std::cout << std::endl;
  return 0;
}
```

A first C++ program

```
#include <iostream>

int main()
{
    // This is a comment!
    std::cout << "Hello, World!";
    std::cout << std::endl;
    return 0;
}
```

A first C++ program

```
#include <iostream>

int main()
{
    // This is a comment!
    std::cout << "Hello, World!";
    std::cout << std::endl;
    return 0;
}
```

A first C++ program

```
#include <iostream>

int main()
{
    // This is a comment!
    std::cout << "Hello, World!";
    std::cout << std::endl;
    return 0;
}
```

A more complicated C++ program

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

int main()
{
  std::cout << "What's your name? ";

  std::string name;
  std::cin >> name;

  std::cout << "Hello, "
            << name
            << std::endl;

  return 0;
}
```

A more complicated C++ program

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

int main()
{
  std::cout << "What's your name? ";

  std::string name;
  std::cin >> name;

  std::cout << "Hello, "
            << name
            << std::endl;

  return 0;
}
```

A more complicated C++ program

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

int main()
{
  std::cout << "What's your name? ";

  std::string name;
  std::cin >> name;

  std::cout << "Hello, "
            << name
            << std::endl;

  return 0;
}
```

A more complicated C++ program

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

int main()
{
  std::cout << "What's your name? ";

  std::string name;
  std::cin >> name;

  std::cout << "Hello, "
            << name
            << std::endl;

  return 0;
}
```

A more complicated C++ program

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

int main()
{
  std::cout << "What's your name? ";

  std::string name;
  std::cin >> name;

  std::cout << "Hello, "
            << name
            << std::endl;

  return 0;
}
```

Different kinds of types

Primitive types (no need to `#include`):

- `int` (1, 3, -5, ...)
- `double` (3.4, 15.29, ...)
- `char` ('a', '\n', 47)
- `bool` (`true`, `false`)

See `Program3.cpp`

Composite types:

- `std::string` ("Hello")
- Pointers (`int*`, `double*`, ...)
- Collections like `std::vector`
- Your own types!

Making something useful!

Celsius to Fahrenheit calculator (typical example)

We use the formula $T_{(\circ F)} = T_{(\circ C)} \times 9/5 + 32$

See `Program4.cpp`

Going both ways

How can we let the user choose whether to convert to Fahrenheit or to Celsius?

Simple: we use an `if` statement:

```
if (choice == "fahrenheit") {
  // Convert fahrenheit to celsius
} else if (choice == "celsius") {
  // Convert celsius to fahrenheit
} else {
  // Something went wrong!
}
```

Small exercise: can you fill in the gaps in `Program5.cpp`?

Repeating things

What if we want to make a program that prints a table like so?

Celsius	Fahrenheit
0	32
10	50
20	68

We could just copy the same line of code over and over...
...or we could just use a `for` loop!

```
for (int i = 0; i < 10; i = i + 1) {
  // do something
}
```

See the code at `Program6.cpp`

Processing files

So far we have been reading and printing to the console using the <<,>> operators

We can use the same technique to read and write from files!

Files in C++ are represented by the `std::fstream` type, which can be found in the `fstream` package. In order to open a file, we need to specify a mode (which can be read-only, write-only or both):

```
#include <fstream>
std::fstream file;
file.open("myFile", in | out);
if (file.is_open) {
   file << "Write something to the file";
   file >> readFromFile;
}
file.close();
```

Example: count the number of occurrences of a word

We want to write a program that:

1. Asks the user for a filename and a word
2. Opens the specified file in read mode
3. Looks at every word in the file, increments a counter if it's a match
4. Prints the number of matches

Question: how do we look at every word in the file?

- We can't use a `for` loop because we don't know how long the file is!
- We use a `while` loop instead
- See `Program7.cpp`

Creating custom functions

Sometimes there is some code we want to repeat in different places

An example: converting from Celsius to Fahrenheit (we have been doing it a lot today!)

We can extract that code into a function like so:

```
double celsiusToFahrenheit(double celsius)
{
  return 32 + celsius * 9.0 / 5.0;
}
```

Function declarations should go *before* the `main` function in a C++ program. See `Program8.cpp` for an example, and `Program9.cpp` for more function stuff.

Time to play around a little!

Let's make a "Guess the Number" game

The file `Program10.cpp` comes with a function `random(int a, int b)` that will pick a number at random from `a` to `b` (inclusive).

You should pick a random number, then have the player try to guess it. After every guess, if the player didn't get it right, give them a hint ("lower" or "higher").

After a certain number of tries, if the player still hasn't guessed the number, they lose!

If you want to try something harder, try to make something like Blackjack with only one suit of cards.
For this, you may want to use the two functions `reset(int a, int b)` and `deal()`

If we have time: vectors and structs

Vectors are like Python lists: they are containers where you can put any number of elements of a certain type.

```
std::vector<int> numbers;

// Inserting at the end
numbers.push_back(10);
// Removing at the end
numbers.pop_back();
// Accessing
numbers[i] = 100;
// Length
numbers.size();
```

Important notes:

- You can't mix and match types, all elements must have the same type
- Passing vectors to functions works, but it can be very slow if the vector is big!

Creating your own types: structs

A struct is a group of values that are bundled together in your program. Good example: complex numbers.

A complex number is a pair $x + yi$, we can represent this by a C++ struct like so:

```
struct Complex {
  double realPart;
  double imaginaryPart;
};
```

You can create a variable of type `Complex` like any other type, and access both fields using dot-syntax.

```
Complex z;
z.realPart = 10.0;
z.imaginaryPart = 5.0;
```

Projet suggestion

A fun little project to try in your free time: a text adventure in the style of Zork!

Some hints:

- Use a struct to represent a room, with fields that include a string description and a vector of the actions that can be performed.
- The rooms should go into a big vector, you can keep track of where the player is with an integer!
- Use a struct to represent an action, with a field for the command that will trigger the action, a field for the kind of action that it is (e.g. moves the player), a string that is printed when the action is performed.

See `Program13.cpp` for a template you can use and try to improve on it!