G52CPP
C++ Programming
Lecture 1

Dr Jason Atkin

This Lecture

- Pre-requisites
- Module Aims and Information
- Module Structure and Assessment
- History of C and C++
- Why Learn C and C++
- C++ vs Java?
- Suggested Course Texts
- ‘C++ Hello World’ (vs ‘C Hello World’)

Pre-requisites

- **G51PRG : Procedural C Programming**
  - You should already know a fair amount about C programming
  - You should understand a considerable number of C functions

- **G51OOP : Object Oriented Java**
  - You should have an understanding of what a class is, what members are, how classes work together

Module Information

- All information is available on the moodle web page:

http://moodle.nottingham.ac.uk/course/view.php?id=26270

Including:

- Lecture slides (a few days before lectures)
- Code samples (when relevant)
- Lab notes (as they are released)
- Coursework information (later)
- Revision information (later)

Module Aims

- Understand and write C++ source code
  - Good for employment prospects
  - **Ultimately, this is a PROGRAMMING module**
- **Understand what your code actually does**
  - **Know something of *how* C++ implements features**
  - **You are using C/C++ for the speed (otherwise choose an easier language) so understand the issues**
- We will see:
  - The similarities and differences between C, C++ and Java (& why)
  - The C++ basics – the most commonly used features
  - What advanced features exist, to look them up if needed
- I will **not** be teaching:
  - Object oriented methodology
  - Object oriented C++
  - How to create C++ programs to satisfy purists

The Iceberg of C++

- We will only look at the top – the common things
- Many other features we will not cover

Five reasons to learn C & C++?

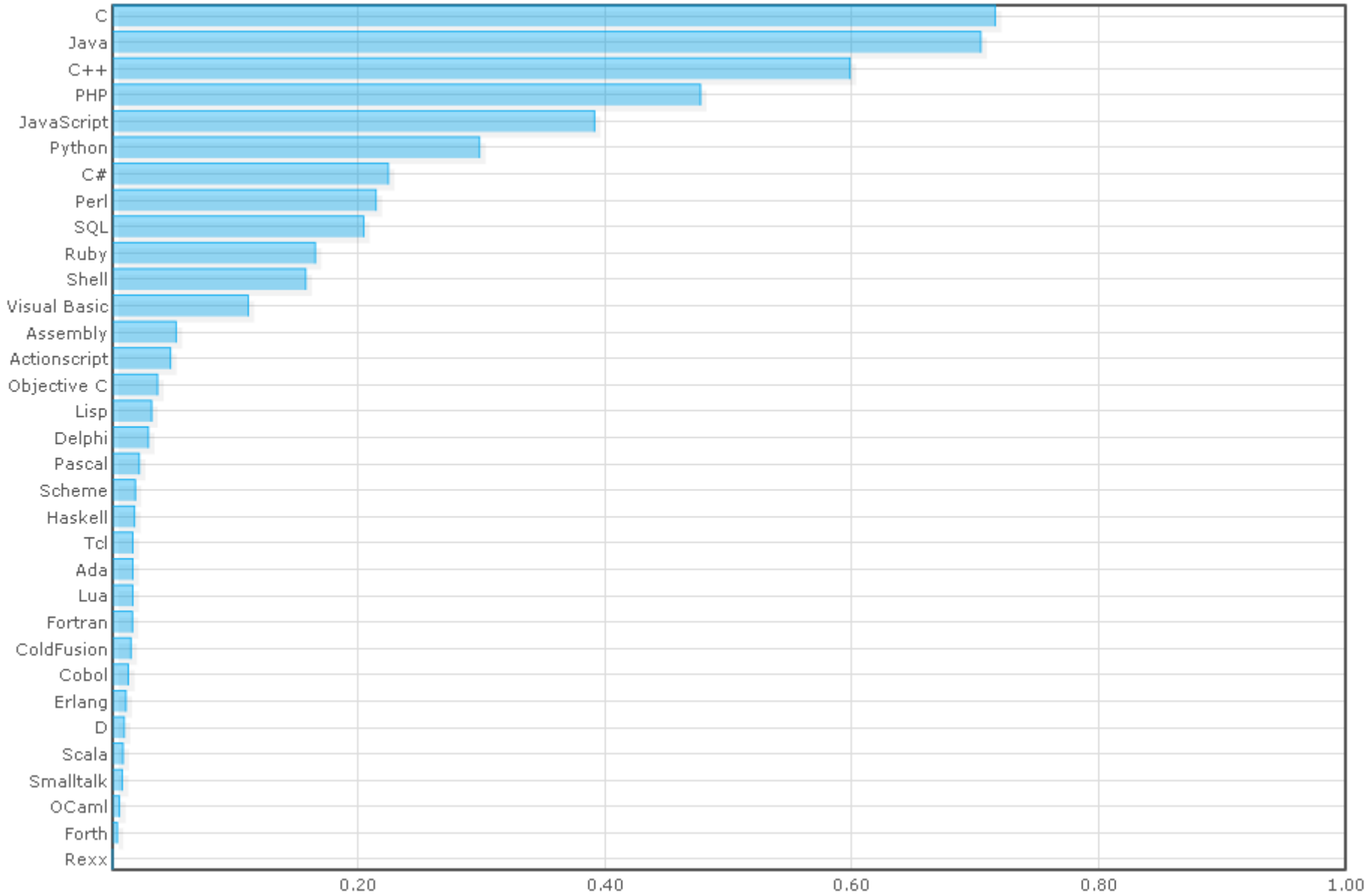
1. (Still) utilised in industry – C even more so
   - Why so popular? (after so long)
2. Choose the appropriate tool for the task
   - Understand the Java/C# vs C/C++ differences
3. More programming practice
   - Much is common across languages
4. Easier to learn: much will be familiar
   - So many similarities to Java
5. Useful for other modules
   - And for 3rd year projects

The history of C/C++

In Bell Labs, ‘B’ language created, based on BCPL

- 1971-1973 : Dennis Ritchie extended ‘B’ to create ‘C’
  - Main features of C developed over this time
- 1973-1980 : New features were added. C needed to be standardised!
- 1979 : Bjarne Stroustrup (Bell labs) extended C to make ‘C with classes’
- 1982 : ‘K&R’ (Kernighan and Ritchie) unofficial C ‘standard’
- **1983 : ‘C with classes’ renamed C++, features still being added**
- 1989 : ANSI standard C (started in 1983!) (=> ISO standard in 1990)
  - Differs in some ways from K&R ‘C’ and is often named ‘**C89**’
  - Together with Amendment 1, forms ‘C’ element of ‘C++’
- **1990s : C++ took centre stage (Standardisation progressing)**
- 1994 : Standard Template Library makes it into the ISO standard C++
  - (Some template implementation arguments ongoing as late as 2003)
- 1995 : Java released by Sun
- **1998 : ISO standard C++ ratified (C++98)**
- 1999 : New version of C standard (C99) (Differs from C++98)
- **2011 : C++11, the latest C++ standard, released, MANY additions**

www.langpop.com, normalised, 2011



http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (Jan 2013)

Position Jan 2013	Position Jan 2012	Delta in Position	Programming Language	Ratings Jan 2013	Delta Jan 2012	Status
1	2	↑	C	17.855%	+0.89%	A
2	1	↓	Java	17.417%	-0.05%	A
3	5	↑↑	Objective-C	10.283%	+3.37%	A
4	4	=	C++	9.140%	+1.09%	A
5	3	↓↓	C#	6.196%	-2.57%	A
6	6	=	PHP	5.546%	-0.16%	A
7	7	=	(Visual) Basic	4.749%	+0.23%	A
8	8	=	Python	4.173%	+0.96%	A
9	9	=	Perl	2.264%	-0.50%	A
10	10	=	JavaScript	1.976%	-0.34%	A
11	12	↑	Ruby	1.775%	+0.34%	A
12	24	↑↑↑↑↑↑↑↑↑↑↑↑	Visual Basic .NET	1.043%	+0.56%	A

PopularitY of Programming Language index PYPL (number tutorials searched for)

Position Jan 2015	Position Jan 2015	Delta in position	Programming language	Share in Jan 2015	Twelve month trends
1	1		Java	25.8 %	-0.5 %
2	2		PHP	12.4 %	-1.7 %
3	3		Python	11.8 %	+1.0 %
4	4		C#	9.9 %	-0.9 %
5	5		C++	8.7 %	-0.1 %
6	6		C	8.2 %	+0.3 %
7	7		Javascript	7.4 %	-0.6 %
8	8		Objective-C	6.7 %	-0.4 %
9			Swift	3.1 %	+3.9 %
10	10		Ruby	2.7 %	+0.1 %
11	9		Visual Basic	2.1 %	-0.6 %

The aims of C and C++

- Even in 2013, C was more popular than C++ and Java
  - Especially for operating systems and device drivers
  - Where layout in memory matters – control needed
- C came first : with specific design aims
  - Ability to write low-level code (e.g. O/S)
  - Speed and efficiency
  - Ease for programmers (rather than non-programmers)
- Cross-platform compilation
  - Compared with Assembly code
  - Not as much as Java
- Why is C still so popular (over C++ and Java)?
  - Control and visibility – don't have the side-effects/simplifications
  - Simpler compiler support / implementation
  - Anything you can do in C++ can be done in C
  - But may need (a lot) more code

Why C++ rather than C?

- **Everything in C++ could be done in C, so why learn C++?**
- C++ gives you higher level concepts
  - Hides complexity
  - Java hides even more and gives no choice but ‘do it my way’
  - C++ keeps the ability to do things as you wish
    - But can be overwhelming in its options ☹ (especially C++11)
- Higher level view is sometimes very useful, when large amounts of code can be reused
  - C++ Class libraries are ideal for a GUI on Windows, OS/X, Linux (then decide appearance/speed vs portability)
- C++ also adds to C a lot of non-OO features
  - e.g. templates, new/delete, operator overloading, references, …
  - Useful for procedural programming as well as O.O.

Object Oriented or Procedural?

- C is procedural (no classes, hard to do OO, but not impossible?)
- In C++:
  - You **CAN** write procedural C++ (OO 'purists' will frown at you)
  - Or you can write object oriented C++
  - Or mix both together (often a bad idea)
- Procedural or O.O. are ***ways of thinking***
  - A lot of communicating objects or 'do this then this then this…'
- Whichever you use: (within a thread)
  - Functions are called and operations are executed one at a time
- Object oriented techniques can hide some complexity (a good thing?)
  - Make it easier to understand a program (?)
  - Make it easier to structure a large program (?)
  - Some facilities hide what is actually happening, to simplify things (bad?)
- We will consider C++ as a language for programming
  - We will **not** look at object oriented design/programming theory

C++ knowledge is respected

- It has been said:

  "If you can do C++, you can do Java and C#"

- What does this mean?
  - Both coverage and complexity
  - C++ is **NOT** easy!
  - You need to understand a lot to 'do' C++

- **Do not expect an easy module!**
- **Expect to have to think!**
  - A good memory will not get you through

C++ vs Java?

(A comparison, not a contest)

What is C++?

Procedural C
Global Functions
File-specific functions
Structs
Pointers (addresses)
Low-level memory access
C Preprocessor

Variables
Arrays
Loops
Conditionals

Classes
- Grouping of related data together
- With associated methods (functions)

'new' for object creation
'delete' for object destruction
Constructors, Destructors
Operator Overloading
Assignment operators
Conversion operators
Inheritance (sub-classing)
Virtual functions & polymorphism
Access control (private/public/protected)

Function Libraries
Standard functions
Custom libraries
O/S functions

Templates
(Generic classes)

Non-C features
e.g. References

Class Libraries
(+templated classes)
Standard library (much C++11)
BOOST ('beta' libraries?)
Platform specific libraries

What about Java?

Procedural C
~~Global Functions~~
~~File-specific functions~~
~~Structs~~
~~Pointers (addresses)~~
~~Low-level memory access~~
~~C Preprocessor~~

Variables
Arrays
Loops
Conditionals

Classes
- Grouping of related data together
- With associated methods (functions)

'new' for object creation
~~'delete' for object destruction~~
Constructors, ~~Destructors~~
~~Operator Overloading~~
~~Assignment operators~~
~~Conversion operators~~ (toString()?)
Inheritance (sub-classing)
(ONLY) Virtual functions & polymorphism
Access control (private/public/protected)

~~**Function Libraries**~~
~~Standard functions~~
~~Custom libraries~~
~~O/S functions~~
Java Native Interface

~~**Templates**~~
'Generics' (weaker)

Non-C features
(ONLY) references

Class Libraries
(Generics ~ C++ templates)
Collections
Networking, Threading, etc
Graphics - standardised

What Sun changed for Java

- Remember: C++ came first
  - Java was based on C++ and the **changes were deliberate**!
- Java is cross-platform
  - Interpreted intermediate byte-code (.class files)
  - Standard cross-platform class libraries
  - Libraries include graphics (AWT, SWING, …), networking, …
  - Platform independent type sizes
  - *Cannot easily take advantage of platform-specific features*
- Java prevents things which are potentially dangerous
  - Pointer arithmetic (but it can be fast)
  - Writing outside arrays (checks take time)
  - Low-level access to memory (dangerous per powerful/quick)
  - Uninitialised data (initialisation takes time)
- Java forces you to use objects
  - Even when it would be quicker not to
- Java does garbage collection for you
  - Safer(?), but may execute slower than freeing memory yourself

My view of C/C++ vs Java

- C++:
  - Power and control: What to do? How to do it?
- Java:
  - "**Do it my way and I'll do more of the work for you**"
  - But it may be less efficient than doing it yourself
  - Some things cannot be done in Java alone (JNI)
- **Java hides many things from you**
  - And decides how you will do things
- **Java prevents you doing some things and checks others**
  - **C++ trusts that you know what you are doing**
  - **If you do not, then you can REALLY break things**
- Q: Do you want/need the power/control of C++?
  - Probably a per-project decision

Which is better? Java or C++?

- What does ‘better’ mean?
- What are you trying to do?
- Do you need the **power and control** that C++ gives you?
- With fewer options, things may seem simpler
  - Potentially harder to make mistakes
  - But you lose the flexibility to optimise
- If you know both, then you have more options (and the basics are very similar)

Platform specific class-libraries

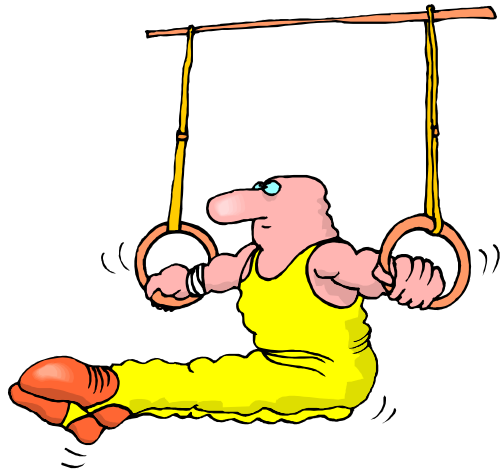
- E.g. Visual studio provides easy support for windows:
  - MDI Child/container windows
  - SDI (Single Document Interface)
  - Dialog based
  - Ribbon bar
  - Tool bar
  - Splitters
  - ActiveX containers and servers
- App-wizard will include these for free
- **Cost: platform dependence**

Module Structure

Organisation

- 3 lectures per week
  - Introducing C and C++ concepts
    - From lecture slides
  - Writing or modifying some C++ code
    - Using a development environment or text editor
    - Prep for labs and coursework
  - Changed from 2 content + one demo lecture
- 1 one-hour lab session
  - Get help and feedback on progress
  - Work on demo programs to learn better …
  - … and coursework later on

What you need to (or should) do



- ***Attend the lectures***
- Try the samples from lectures
- Attend the lab sessions
  - I suggest doing the exercises
  - Come with questions
  - Work on it outside of labs
- Read books & online web sites about C/C++
- Try your own sample programs
  - From books or online
- Change existing code, observe the effects

Feedback

- How do you know how well you are doing?

Feedback

- How do you know how well you are doing?

- Brilliant automated feedback device: the compiler
  - Check compilation warnings as well as errors!

- Utilise lab sessions please
  - Many helpers have done this module before, from your position
  - Come along with issues/questions ready

Assessment

Course Assessment

- Course assessment is by:
- Coursework (40%)
  - Due in straight after Easter break (the Tuesday)
- Exam (60%)
  - Answer Q1 plus one of Q2 or Q3
  - Q1 is understanding and writing C/C++ code
    - More common features, e.g. pointers, references, constructors, inheritance, etc
    - i.e. The things that everyone should know and will use a lot
  - Q2/Q3 are more specialised (complex?) issues
    - E.g. exceptions, v-tables, multiple inheritance, C++ container classes, lambda functions (C++11 only) etc

Exam Questions

- I will assume the following are valid for exam questions:
  - Things covered in the lectures
    - Even if not on the lecture slides!
    - I will put any extra code on the web page
  - Things in the samples or lab notes
    - Especially the first few labs
  - The basic C/C++ language constructs
    - You were introduced to many of these in G51PRG
    - Operators, loops, conditionals, etc
  - The common C library functions (part of C++)
    - e.g. input/output functions, string functions, printf, …
    - Even where we have not covered them in detail

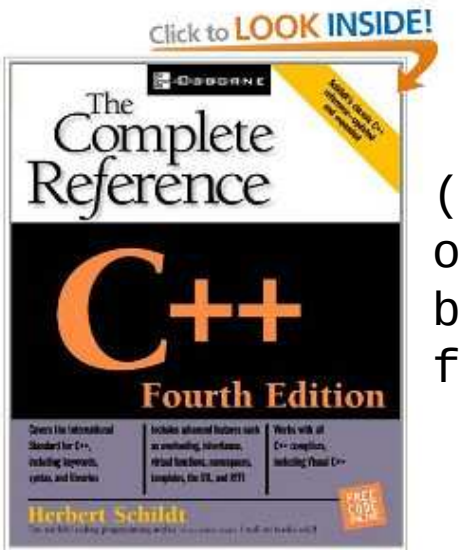
Formal Coursework : 40%

- C++ programming
- A simple graphical program using the SDL multimedia library and the supplied framework
- Deliberately not a framework which you will already know!
- You need to provide a number of features
  - A lot of flexibility for you to choose how to meet the requirements (helps me catch plagiarism too)
- Hall of fame for previous years can be found here: http://www.cs.nott.ac.uk/~jaa/cpp/CPPHallOfFame/halloffame.html
  - Download and try some of the games
  - Look at the variety of things produced
  - Think about what you would like to do
- Some changes from last year (version and requirements)

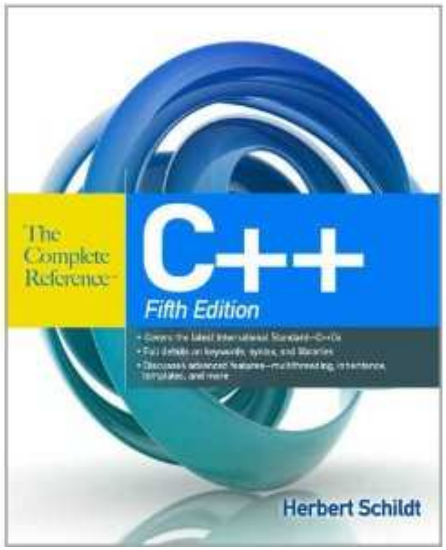
Course Text?

Semi-recommended text

- The Complete Reference: C++, Fourth Edition by Herbert Schildt
  - Similar structure to this module (C then C++)
  - A **reference**, not a tutorial!



(Images courtesy of amazon, but books are available from many places)



You are better off using the time to try coding though!

Other Texts (1)

- **Many other C++ books in the library**
  - But most books now introduce classes from the start (so you think in an OO way)
  - E.g. ‘C++ How to Program’, Deitel and Deitel
  - **Many** other books in the Jubilee library
- **Online references are probably the most help**
  - **E.g. http://www.cplusplus.com/doc/tutorial/**
  - **To look up examples or explanations**

Other Texts (2)

- '**The C++ Programming Language**',
  by Bjarne Stroustrup
  - The definitive book on the language
  - *but not a tutorial* (a technical reference)
  - Only 4th edition covers C++11!

- '**Effective C++**' and '**More Effective C++**',
  by Scott Meyers
  - Explain many confusing elements
  - Ideal for understanding 'why' as well as 'how'
  - NOT an introduction to C++
  - Written before C++11

“Hello World”

A simple C++ (and C) program

The “Hello World” Program

```
#include <stdio.h> /* C file */

int main(int argc, char* argv[])
{
  printf("Hello world!\n");
  return 0;
}
```

C version

```
#include <cstdio> /* C++ file */

int main(int argc, char* argv[])
{
  printf("Hello world!\n");
  return 0;
}
```

C++ version

Next lecture

- The basics, including compilation, pre-processor and using multiple files
- **NO LABS THIS WEEK**
- **IMPORTANT:**
  - **If you can’t remember the basics of C pointers, make sure that you attend the Thursday G52OSC lecture about pointers before the Friday G52CPP lecture**