

# G52CPP

## C++ Programming

### Lecture 1

Dr Jason Atkin

E-Mail: [jaa@cs.nott.ac.uk](mailto:jaa@cs.nott.ac.uk)

# This Lecture

- Pre-requisites
- Module Aims and Information
- Suggested Course Texts
- Module Structure and Assessment
- History of C and C++
- Why Learn C and C++
- C++ vs Java?
- '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 Aims

# Aims: I want you to ...

- Be able to understand and write C/C++ source code
  - Good for employment prospects
  - **Ultimately, this is a PROGRAMMING module**
- Know what you can do in C++
  - Be able to look up 'how' in a job if necessary
- **Understand what your code actually does**
  - **Know something of *how* C++ implements features**
- Know the similarities and differences between C, C++ and Java
  - Understand why Java changed things
- Practise programming (will help with Java and C too)
- I will **not** be teaching:
  - Object oriented methodology
  - Object oriented C++
  - How to create C++ programs to satisfy purists

# We will start with procedural C++

**Builds on G51PRG**

**Understand what is  
happening  
inside/underneath C++**

Rather than hiding it

If you understand what it is  
doing, you can diagnose  
and fix your program when  
it goes wrong

Otherwise you need to ask  
for help



# The Iceberg of C++

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



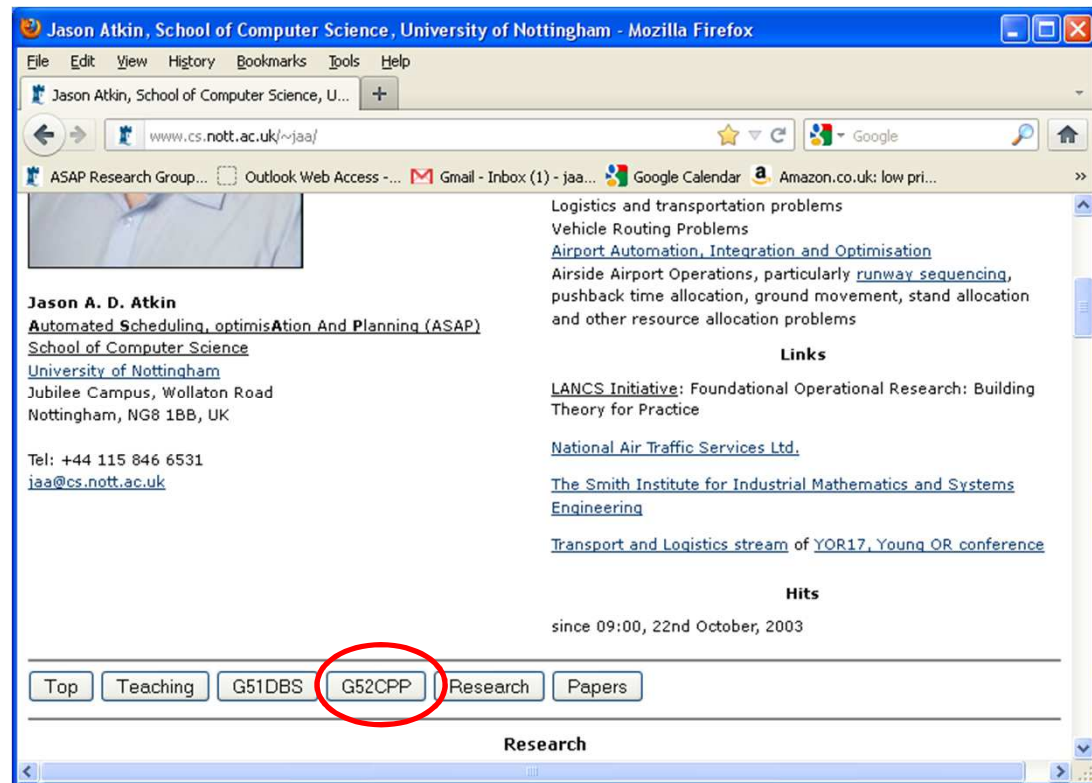
- We will also ignore object oriented theory ('how' not 'why')

# Module Information

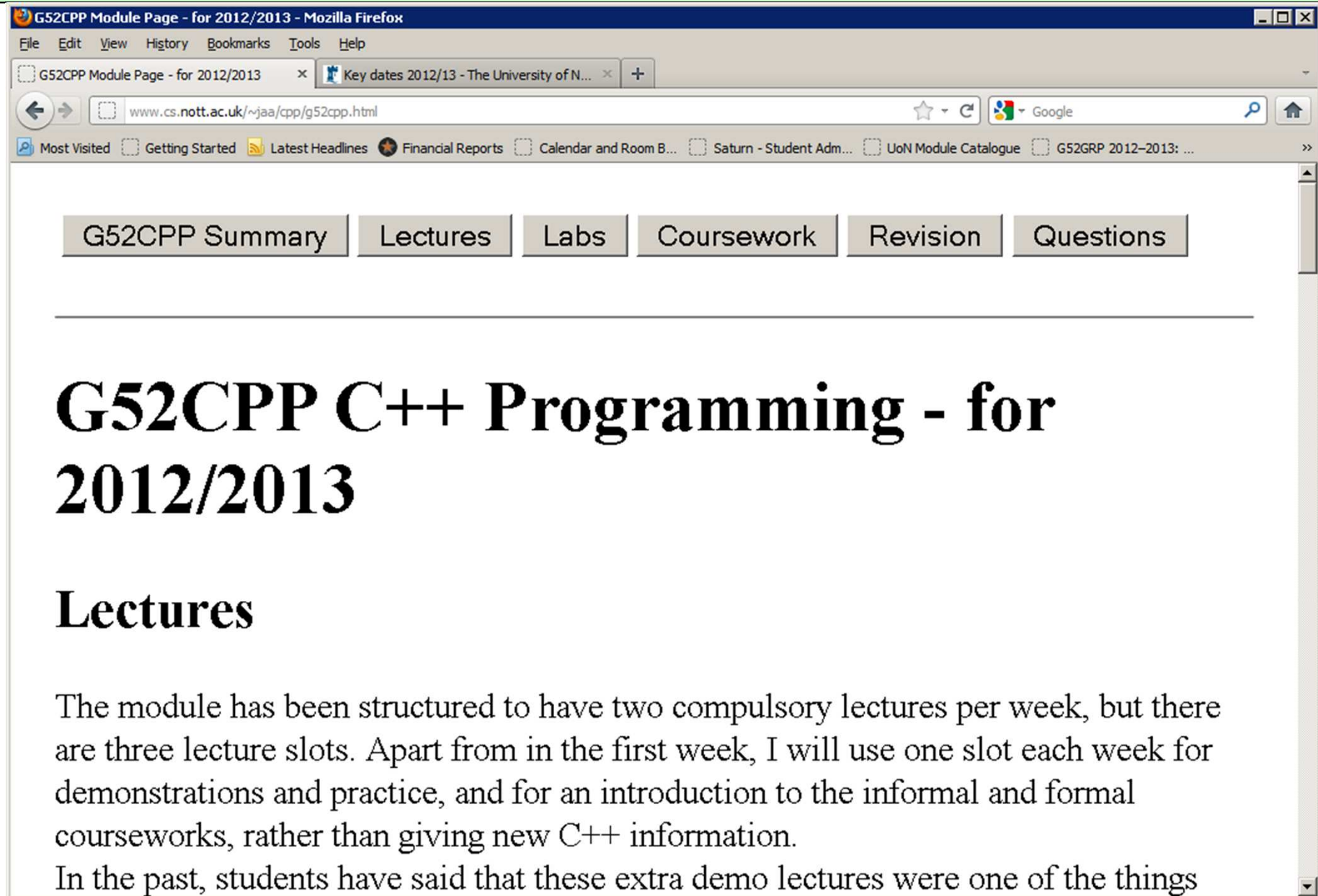


# Information

- Lecture slides, sample code and lab notes will be on the module web page:
  - <http://www.cs.nott.ac.uk/~jaa/cpp/g52cpp.html>
- Or google me (Jason Atkins)
- And scroll down to the buttons on my homepage



# Main Web Page



The screenshot shows a Mozilla Firefox browser window with the title "G52CPP Module Page - for 2012/2013 - Mozilla Firefox". The address bar displays "www.cs.nott.ac.uk/~jaa/cpp/g52cpp.html". The page features a navigation bar with buttons for "G52CPP Summary", "Lectures", "Labs", "Coursework", "Revision", and "Questions". Below this, the main heading reads "G52CPP C++ Programming - for 2012/2013". Under the heading, the section "Lectures" is displayed. The text describes the module structure: "The module has been structured to have two compulsory lectures per week, but there are three lecture slots. Apart from in the first week, I will use one slot each week for demonstrations and practice, and for an introduction to the informal and formal courseworks, rather than giving new C++ information. In the past, students have said that these extra demo lectures were one of the things".

G52CPP Module Page - for 2012/2013 - Mozilla Firefox

File Edit View History Bookmarks Tools Help

G52CPP Module Page - for 2012/2013 x Key dates 2012/13 - The University of N... x +

www.cs.nott.ac.uk/~jaa/cpp/g52cpp.html

Most Visited Getting Started Latest Headlines Financial Reports Calendar and Room B... Saturn - Student Adm... UoN Module Catalogue G52GRP 2012-2013: ...

G52CPP Summary Lectures Labs Coursework Revision Questions

---

## G52CPP C++ Programming - for 2012/2013

### Lectures

The module has been structured to have two compulsory lectures per week, but there are three lecture slots. Apart from in the first week, I will use one slot each week for demonstrations and practice, and for an introduction to the informal and formal courseworks, rather than giving new C++ information.

In the past, students have said that these extra demo lectures were one of the things

# Lecture Slides and Information

G52CPP Lectures - Mozilla Firefox

File Edit View History Bookmarks Tools Help

G52CPP Lectures x Key dates 2012/13 - The University of N... x +

www.cs.nott.ac.uk/~jja/cpp/lectures.html

Most Visited Getting Started Latest Headlines Financial Reports Calendar and Room B... Saturn - Student Adm... UoN Module Catalogue G52GRP 2012-2013: ...

## Lecture slides:

Note: Any code from demo lectures will appear on the Labs pages, since it fits better there, with the lab notes associated with it.

Lecture and Date	Link to slides	Subject
Lecture 1, Thurs 31st Jan	<a href="#">Preliminary slides</a>	Introduction and Overview
Lecture 2, Fri 1st Feb	To appear	Data types and casting, operators, Pointers
Lecture 3, Fri 1st Feb	To appear	Pointers, arrays and strings
*** No lab on Friday 1st February, A32 ***		
Lecture 4, Thur 7th Feb	To appear	Pointer arithmetic, functions and globals
Lecture 5, Fri 8th Feb	To appear	The stack, variable shadowing and variable lifetime
*** Labs start Fri 8th February, A32 ***		
Lecture 6, Thur 14th Feb	To appear	Structs, unions and sizes
Lecture 7, Fri 15th Feb	To appear	Dynamic memory allocation and linked lists
*** Lab 2 is Friday 15th February, A32 ***		
Lecture 8, Thur 21st Feb	To appear	Const, multiple files and the pre-processor
Lecture 9, Fri 22nd Feb	To appear	Classes

# Lecture Slides and Information

G52CPP Lectures - Mozilla Firefox

File Edit View History Bookmarks Tools Help

G52CPP Lectures x Key dates 2012/13 - The University of N... x +

www.cs.nott.ac.uk/~jja/cpp/lectures.html

Most Visited Getting Started Latest Headlines Financial Reports Calendar and Room B... Saturn - Student Adm... UoN Module Catalogue G52GRP 2012-2013: ...

## Lecture slides:

Note: Any code from demo lectures will appear on the Labs pages, since it fits better there, with the lab notes associated.

**WARNING:** If lecture attendance is low I will start to remove parts of the slides on the web – so that you have to fill them in

Lecture 1		
Lecture 2		
Lecture 3		
*** No lab		
Lecture 4		
Lecture 5		
*** Labs		
Lecture 6		
Lecture 7, Fri 15th Feb	To appear	Dynamic memory allocation and linked lists
*** Lab 2 is Friday 15th February, A32 ***		
Lecture 8, Thur 21st Feb	To appear	Const, multiple files and the pre-processor
Lecture 9, Fri 22nd Feb	To appear	Classes



# Lab Notes and Demo Lectures

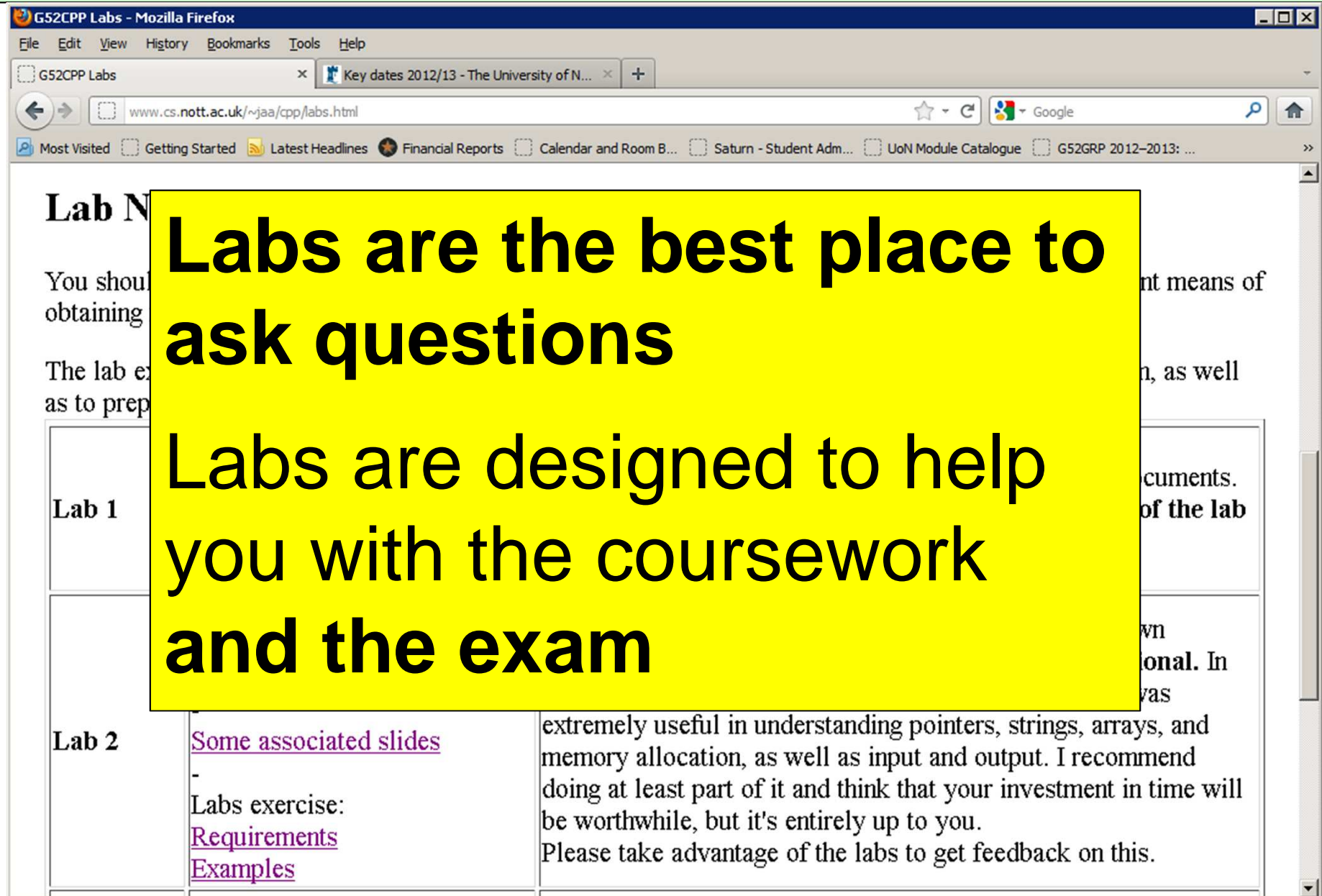
**Lab Notes and Demo Lecture details**

You should use the labs to practice and get **feedback** from the lab assistants or me. This is an important means of obtaining feedback on your progress and you should not ignore it.

The lab exercises have been designed to teach you important concepts which you will need in the exam, as well as to prepare you for the formal coursework.

<b>Lab 1</b>	<a href="#">Introduction: Getting started</a> - <a href="#">Exercises for Lab 1</a> - <a href="#">Some associated slides.</a>	Please use the first lab session to work through these documents. <b>There is a short program for you to write at the end of the lab notes document.</b>
<b>Lab 2</b>	<a href="#">Input/Output examples</a> - <a href="#">Code sample : io_demo.cpp</a> - <a href="#">Some associated slides</a> - Labs exercise: <a href="#">Requirements</a> <a href="#">Examples</a>	Most of this lab session revolves around writing your own program for a Hangman game. <b>This is ENTIRELY optional.</b> In previous years, students said that this kind of exercise was extremely useful in understanding pointers, strings, arrays, and memory allocation, as well as input and output. I recommend doing at least part of it and think that your investment in time will be worthwhile, but it's entirely up to you. Please take advantage of the labs to get feedback on this.

# Lab Notes and Demo Lectures



The screenshot shows a Mozilla Firefox browser window with the title "G52CPP Labs - Mozilla Firefox". The address bar displays "www.cs.nott.ac.uk/~jaa/cpp/labs.html". The page content is partially obscured by a large yellow text box. The text box contains the following text:

**Labs are the best place to ask questions**

**Labs are designed to help you with the coursework and the exam**

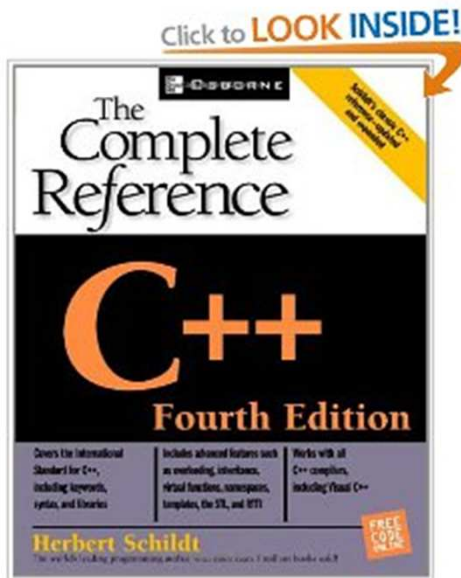
The background page shows a table with lab information. The visible parts of the table are:

Lab N	
Lab 1	
Lab 2	<p><a href="#">Some associated slides</a></p> <p>Labs exercise:</p> <p><a href="#">Requirements</a></p> <p><a href="#">Examples</a></p> <p>extremely useful in understanding pointers, strings, arrays, and memory allocation, as well as input and output. I recommend doing at least part of it and think that your investment in time will be worthwhile, but it's entirely up to you. Please take advantage of the labs to get feedback on this.</p>

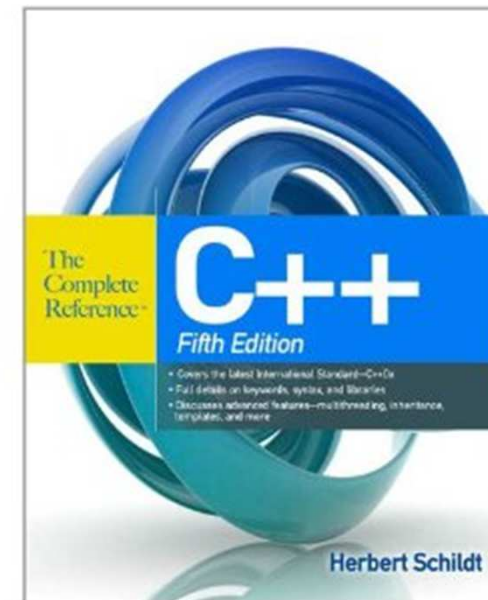
Course Text

# Recommended Course 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 available from many places.)





# 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** books in the Jubilee library
- **Other online references**
  - E.g. <http://www.cplusplus.com/doc/tutorial/>

# Other Texts (2)

- **‘The C++ Programming Language’**,  
by Bjarne Stroustrup
  - The definitive book on the language
  - *but not a tutorial* (a technical reference)
- **‘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++

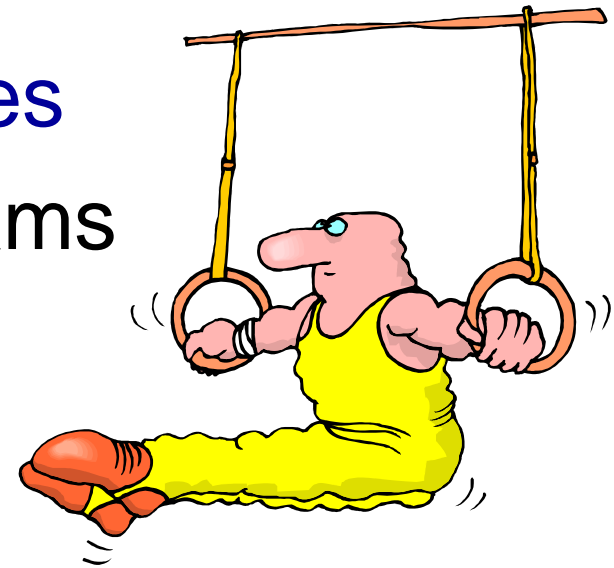
# Module Structure

# Organisation

- 2 'new material' lectures per week
  - Introducing C and C++ concepts
  - **3 in week 1** – things you should already know
- 1 demo lecture – in a lecture room/slot
  - Practical demonstrations
  - Usually writing/examining code, no slides
  - Introduces the coursework
- 1 two-hour lab session
  - Get help and feedback on progress

# What you need to (or should) do

- ***Attend the lectures***
- Read books & online web sites about C/C++
- Attend the lab sessions
  - I suggest doing the exercises
- Try the samples from lectures
- 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 session helpers please
  - All 3 this year have done this module before!
  - 2 hour session each week

# Assessment



# Course Assessment

- Course assessment is by coursework (40%) and exam (60%)
- 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!
    - A lot of flexibility for you to choose how to meet the requirements



# Exam Questions

- I will assume the following are valid for exam questions:
  - Things covered in the lectures
    - Even if not on the lecture slides!
  - Things in the samples or lab notes
    - **Especially the first two lab notes**
  - 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

# Coursework

- Write a program using the supplied framework (which does a lot of the work)
- You need to provide a number of features
- 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

C and C++

# 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++)


**We will consider C++, G51PRG covered parts of C89**


# Five reasons to learn C & C++?

 (Still) utilised in industry – C even more so  
– Why so popular? (after so long)

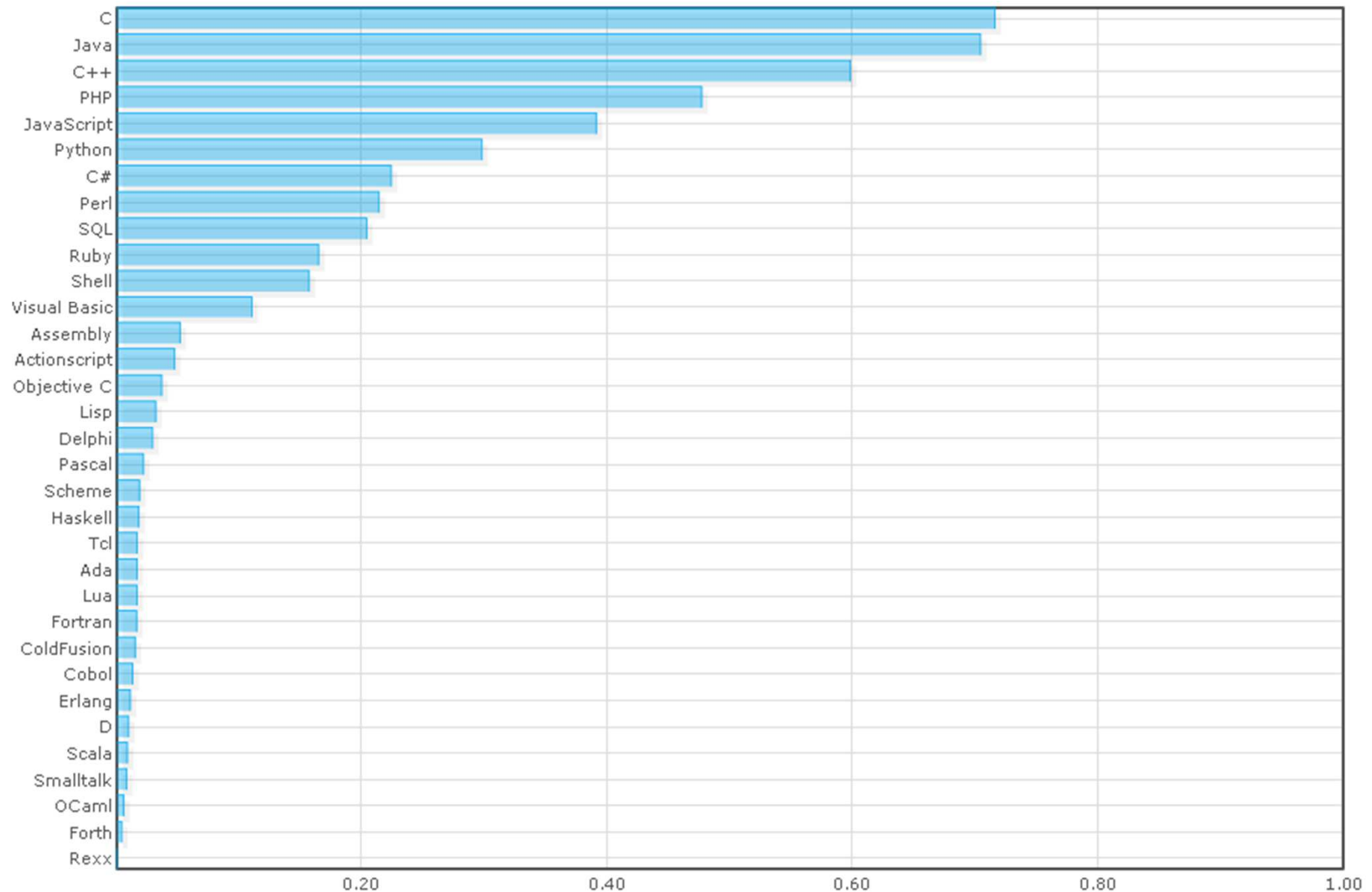
 Choose the appropriate tool for the task  
– Understand the Java/C# vs C/C++ differences

 More programming practice  
– Much is common across languages

 Much is VERY similar to Java and C#  
– Easier to learn: much will be familiar to you

 Useful for other modules  
– And for 3<sup>rd</sup> year projects

# [www.langpop.com](http://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



# The aims of C and C++

- Even in 2011, 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
  - Anything you can do in C++ can be done in C
  - But may need more code

# Why C++ rather than C?

- **Since everything in C++ could be done in C, 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
- 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)
- C++ will let you do **either**
  - 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, **not** at object oriented design/programming

# C++ knowledge is respected

- It has been said:  
“If you can do C++, you can do Java and C#”
- What does this mean?
  - 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?

# 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 + BOOST  
Custom 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

(Standardised)  
Collections  
Networking  
Graphics

# 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
  - ActiveX servers
- App-wizard will include these for free
- **Cost: platform dependent**



# What Sun changed for Java

- Remember: C++ came first
  - The Java 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 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)
  - 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<sup>41</sup>

# 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**
- Do you want/need the power/control of C++?

# 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)



“Hello World”

A simple C++ (and C) program

Since you have had 2 semesters to  
forget what you did in G51PRG

# 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

- C++ data types
  - Sizes of types
  - C and C++ types
  - Two new C++-only types
- Type casting
- Operators
- **Pointers (introduction)**