SUPA Graduate C++ Course

Lecture 1

S. Allwood-Spiers
University of Glasgow

With thanks to W. H. Bell (now at Geneva University)

Course Overview

- Lecture 1
  - Basic C++ syntax
  - Header Files
- Lecture 2
  - File Input/Output
  - Classes and Objects
  - Object Communication
  - Operator Overloading
- Lecture 3
  - Inheritance
  - Polymorphism and Interfaces
  - Templates and STL algorithms
- Lecture 4
  - Applications and ROOT

Assessment

- 3 problem sheets:
  - Problem Sheet 1: Problem 1 only. Deadline 2nd November
  - Problem Sheet 2: 3 questions. Deadline 16th November
  - Problem Sheet 3: 3 questions. Deadline 30th November

Ideally attempt problems before the tutorial, and get help with any issues in tutorial.

Lecture Overview

- Introduction
  - Foreword
  - Programming Methodology
- Basic C/C++ Syntax
  - Simple types and operators
  - Functions
  - Loops
  - Conditional Statements
  - Pointers and arrays
- Header files

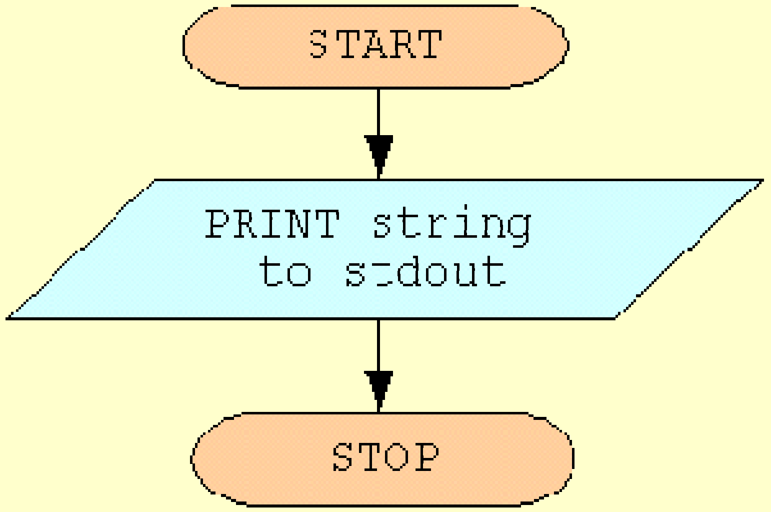
Foreword

- Form a plan of the program needed before writing any C++.
  - Use a flowchart or pseudo-code
  - Think through the implementation
- A little planning at the beginning can save a lot of time later on.
  - This is especially true of Object Orientated languages, of which C++ is the worst in this respect.

Programming Methodology

1. Requirements
2. Design
3. Implementation
4. Documentation

A first C++ program



```
PRINT a string
RETURN 0 to the Operating System
```

http://en.wikipedia.org/wiki/Flow_chart
http://en.wikipedia.org/wiki/Pseudocode

A first C++ program

- A C++ program has a main() function from which the application will start running.
- main() must return an int.
- the return statement in main returns control to the operating system
- Statements end with a semicolon.

```
int main() {

  return 0;
}
```

Extract from InTheBeginning.cc

A first C++ program

includes a standard header file for input and output

```
#include <iostream>
```

standard output stream: *cout* in namespace *std*

```
int main() {
  std::cout << "In the beginning..." << std::endl;
  return 0;
}
```

Shift operator << puts the string into *cout*

extract from InTheBeginning.cc

A first C++ program

```
/* S. Allwood-Spiers
** A very simple C++ program to print one line to
** the standard out
*/

#include <iostream>

using namespace std;

int main() {
  cout << "In the beginning..." << endl;
  return 0;
}
```

comments: ignored by the compiler.
// inline comment, rest of line is ignored
/* */ multiline comment.

InTheBeginning.cc

Compiling C++ on LINUX

- Using the GNU C++ Compiler

```
g++ -o executable filename.cc
```

```
g++ -c file1.cc
g++ -c file2.cc
g++ file1.o file2.o -o executable
```

- Documentation
  - Man pages   man g++
  - Info pages   info gcc
  - Web pages   http://gcc.gnu.org/onlinedocs/

Types

C++ has some predefined types:

Name	Description	Size*	Range*
char	Character	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value.	1byte	false or true (zero or non-zero)
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

* System dependent

Variable Declarations and Definitions

- Declaring a variable reserves some memory for it.

```
int i; //declare a variable of type int with identifier (name )i
```

- Variable names must start with a letter or _ (though I recommend avoiding _). C++ is case-sensitive.
- Some keywords are reserved (see http://www.cppreference.com/wiki/keywords/start).
- Initialisation can occur in same step as declaration:

```
int i; //declaration
i=3;    //assignment
```

Assignment operator

```
int j = 5; //declaration and initialisation
```

Mathematical Operators

Operator	**Meaning**
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulus
i++	i=i+1 (after operation)
++i	i=i+1 (before operation)
i+=7	i=i+7
--i	i=i-1 (before operation)
i--	i=i-1(after operation)

- Other basic mathematical functions can be found in <cmath> and <math.h>

```
j=i++;  means j=i then i=i+1
j=++i; means i=i+1 then j=i
```

Example

```
#include <iostream>

using namespace std;

int main() {
  int myNum = 3;
  int j = myNum++;

  cout << “myNum = ” << myNum << “, j= ” << j << endl;

  int myNumCubed = myNum*myNum*myNum;

  cout << “The cube of “ << myNum << “ is “ << myNumCubed <<endl;

  return 0;

}
```

Extract from ex2/SimpleMaths.cc

Functions

A function is declared with

- a return type,
- a name,
- the type of parameters to be passed to it

```
int cubeNumber(int);

int main() {
  int myNum = 3;
     …
  int myNumCubed = cubeNumber(myNum);
  cout << “The cube of “ << myNum << “ is “ << myNumCubed <<endl;
  return 0;
}

int cubeNumber(int i) {
  return (i*i*i);
}
```

Extract from ex3/SimpleFunctions.cc

Functions

```
void numFingers(int);
void pickColour(void);
bool quitTime(void);

int main() {
  ...
    numFingers(3);
    pickColour();
  ...
  return 0;
}

void numFingers(int) {
  ...
}

void pickColour(void) {
  ...
}
```

Extract from ex4/StdioTests.cc

Relational and Logic Operators

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

$\}$ For comparing two expressions or variables. Result of the comparison is a boolean (true or false).

Operator	Meaning
!	Not

$\}$ Reverse the result of a boolean

Operator	Meaning
&&	And
||	Or

$\}$ For comparing two booleans

Note:

- Single & and | are bitwise operators
- A single = is an assignment operator.

Loops

```
int main() {
  ...
  for(int i=0;i<4;i++) {
    ...
  }
}
...
```

Extract from ex6/Pointers.cc

- statement in this for loop will be executed 4 times,

  when i=0, 1, 2 and 3. i is then set to 4 and loop exits.

Loops

```
int main() {
  do {
    ...
  } while (!quitTime()); // Loop until ready to quit.
  return 0;
}
```

Extract from ex4/StdioTests.cc

- iterates 1 or more times, while the condition is true.
- Can just use `while(condition){}` : iterates 0 or more times.

Conditional Statements

```
void numFingers(void) {
  int fingers;
  ...
  if(fingers==3) {
    cout << "Correct!" << endl;
  }
  else if(fingers>10 || fingers<0) {
    cout << "That is not possible with two hands!"
         << endl;
  }
  else {
    cout << "Wrong.  Try again." << endl;
  }
}
```

Extract from ex4/StdioTests.cc

Conditional Statements

```
void pickColour(void) {
  char colourFlag;
  ...
  switch (colourFlag) {
  case 'y' : cout << "Custard..." << endl; break;
  case 'g' : cout << "Green..." << endl; break;
  case 'b' : cout << "As..." << endl; break;
  case 'r' : cout << "Fast..." << endl; break;
  default : cout << "That..." << endl; break;
  }

}
```

Extract from ex4/StdioTests.cc

- Faster than `if`, `else if`, `else` for some operations.

Scope

- The region in which a variable is valid is known as its scope.

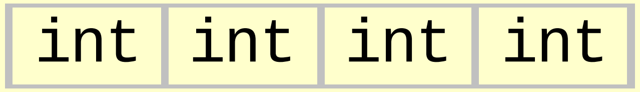
```
int main() {
  int j = 3; // j is in scope

  ...
  for(int i=0;i<2;i++) { // i is in scope
     //j is in scope
     //i is in scope
    ...
  } // i is out of scope

  //j is in scope

} //j is out of scope
```

Extract from ex5/ScopeTest.cc

Arrays

- An array is a sequential block of memory, the size of which depends on the type and the number of elements.

```
int arr[4];
```



- First element: arr[0], last element: arr[3]
- Declaring an array causes memory to be assigned but does not zero elements.

Pointers

- A pointer points to a memory address
  - Initialise with memory address
  - Use to access value in memory address
- Unlike a variable declaration pointer declarations do not cause memory to be assigned
  - Uninitialised pointers can be dangerous.

Pointers and Arrays

```
int main() {
  ...
  int v[] = {1,2,3,4};
  int *pv = &v[0];

  cout << endl;
  for(int i=0;i<4;i++) {
    cout << "v[" << i << "]=" << *pv <<
      "\t &v[" << i << "]=" << pv << endl;
    pv++;
  }
}
```

Extract from ex6/Pointers.cc

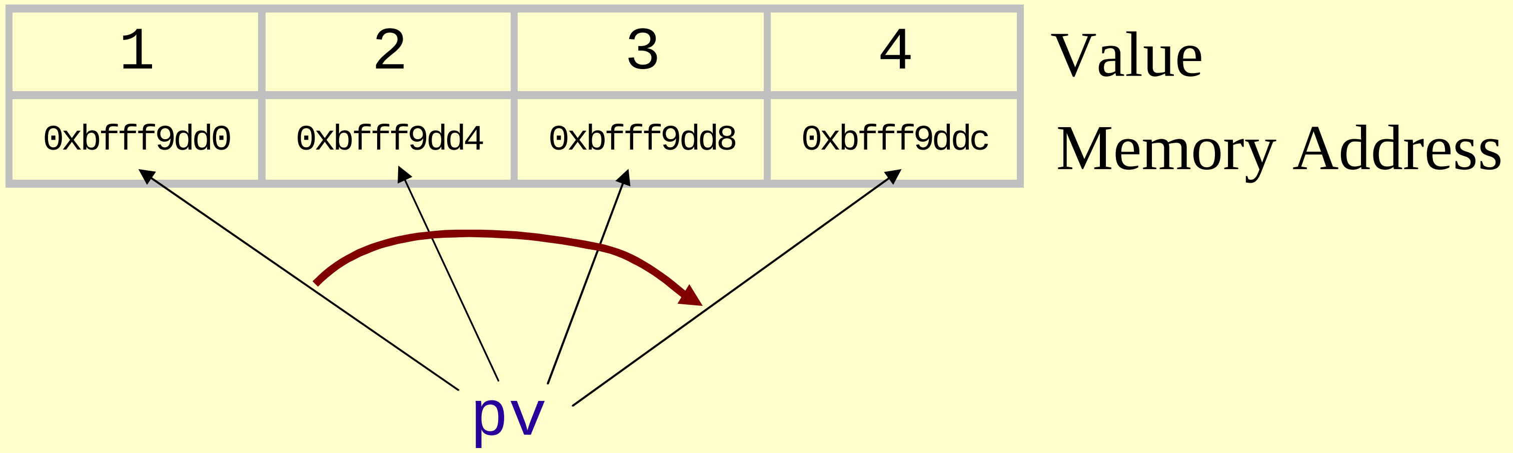
```
int *pv;
pv = &v[0];
```

Alternative code

- pv is a pointer to an int, initialised with the address of the first element of v[] .
- To access the value: *pv

Pointers and Arrays

- Incrementing the pointer pv causes it to point at the next memory location



- The value stored in the given memory address can be accessed with *pv

Pointers and Functions

```
void fun(int, int *);

int main() {
  int np = 1, p = 1;

  cout << "Before fun(): np=" << np << " p=" << p << endl;
  fun(np, &p);
  cout << "After fun(): np=" << np << " p=" << p << endl;
  ...
}

void fun(int np, int *p) {
  np = 2;
  *p = 2;
}
```

Extract from ex6/Pointers.cc

Pointers and Functions

- Passing an array name to a function passes a pointer to the first element
- Objects passed into functions behave in a similar way to simple variables in the given example
  - If changes made within a function are needed after the function has executed Pointers or References should be used.

Header Files

- Can contain:
  - Pre-definition of functions
  - Class declarations (lecture 2)
  - Variable declaration
- Processed during pre-compilation.
  - Pre-compiler has its own syntax

Header Files

Prevent multiple declarations

```
#ifndef STDIO_TESTS_HH
#define STDIO_TESTS_HH

void numFingers(int);
void pickColour(void);
bool quitTime(void);

#endif
```

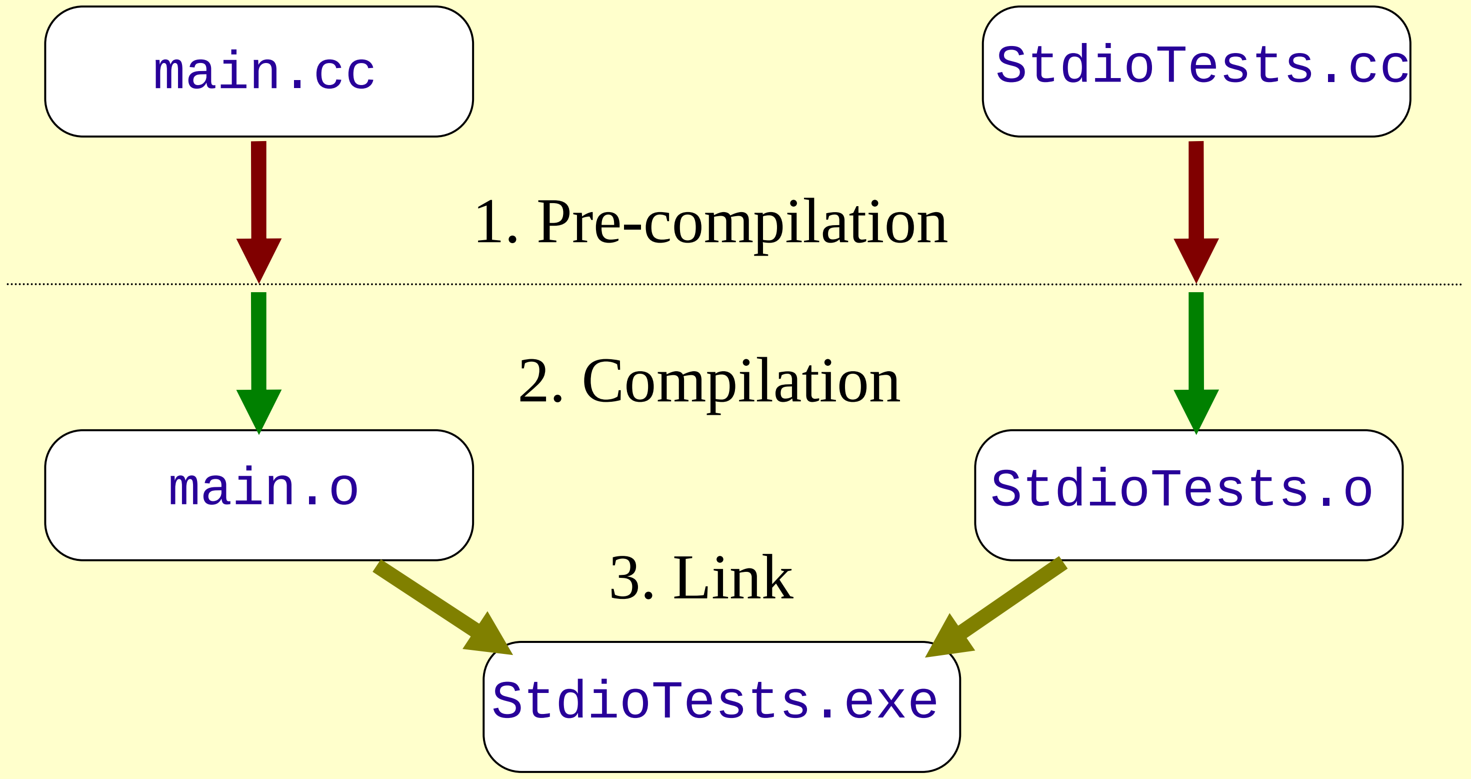
Extract from ex7/StdioTests.hh

```
...
#include “StdioTests.hh"
...

int main() {
  ...
    pickColour();
  ...
}
```

Extract from ex7/main.cc

Must be in the include path

Building an Executable



- When linking with `g++`, `ld` is used
- The `ld` command line depends on which gnu compiler is used

Examples and Problems

- Tutorial session: Now, in room 320.
- For work outside of tutorials:
  - Download session 1 examples from
  - http://my.supa.ac.uk/course/view.php?id=17
  - Build and test examples
  - Attempt problem1 of 1st tutorial sheet.

Extras (to be covered in lecture 2)

Command Line Arguments

Number of arguments given to the command line

```
int main(int argc, char *argv[]) {
  cout << "argc=" << argc
       << " (argc => size of argv array)" << endl;
  for(int i=0;i<argc;i++) {
    cout << "argv[" << i << "]=" << argv[i] << endl;
  }
  return 0;
}
```

Extract from CommandLine.cc

```
./CommandLine.exe arg1 arg2 arg3
```

argv[0] argv[1] argv[2] argv[3]

Streams

- A stream is an object that characters can be inserted to (e.g. cout) or extracted from (e.g. cin).
- Streams provide a uniform basis for input and output independent of device
- Streams allow access to i/o devices, e.g.:
  - files stored on a hard drive
  - the terminal or console
  - a printer
  - a database

Output File Streams

```
#include <fstream>

using namespace std;

void fileWrite(char *filename) {
  ofstream file(filename);

  for(int i=1;i<=20;i++) {
    file << i;
    if(i%5==0) {
      file << endl;
    }
    else {
      file << " ";
    }
  }
  file.close();
}
```

Extract from FileIO.cc

Input File Streams

```
#include <fstream>
...
void fileRead(char *filename) {
  int i;
  ifstream file(filename);

  if(!file) {
    cerr << "Error: could not open " << filename << endl;
  }
  else {
    cout << "Reading file " << filename << endl;
    while(!file.eof()) {
      file >> i;
      cout << i << " ";
      if(i%5==0) cout << endl;
    }
    file.close();
  }
}
```

Extract from FileIO.cc

Make

- A useful tool for building executables and libraries
- Documentation:
  - Man pages   man make
  - Info pages   info make
  - Web pages
    http://www.gnu.org/software/make/manual/make.html

Make Files

```
# S. Allwood-Spiers
# A Makefile to build FileIO.exe

CC=g++
TARGET=FileIO
OBJECTS=main.o FileIO.o

$(TARGET).exe: $(OBJECTS)
	@echo "**"
	@echo "** Linking Executable"
	@echo "**"
	$(CC) $(OBJECTS) -o $(TARGET).exe

clean:
	@rm -f *.o *~

veryclean: clean
	@rm -f $(TARGET).exe
```

```
%.o: %.cc
	@echo "**"
	@echo "** Compiling C++ Source"
	@echo "**"
	$(CC) -c $(INCFLAGS) $<
```

- Provided the file is called Makefile, just type make to build
- make without any arguments builds the default target

References

- References: Similar to pointers in many ways, but different syntax and less flexible. Declare with <type> &<name>: e.g.

```
int myVar=1;
int &refToVar = myVar; //refToVar is a reference to myVar.
```

- Must be initialised at creation, and cannot be changed to refer to another object.
- Use a reference as if it was a value: Value accessed by `refToVar`, address accessed by `&refToVar`.
- When used as an argument to functions, the caller does not need to explicitly say they are using a reference.

```
void fun(int nr, int &r);  //function fun expects an int and
                          // a reference to an int as arguments.
int main() {
  int nr=1, r=1; //nr and r are both ints.

  fun(nr, r); // nr will be passed by value,
              // r will be passed by reference.
}
```

Extract from section1: ex8/references.cc