SUPA Graduate C++ Course
Lecture 1

T. Doherty

University of Glasgow

Course Overview

- Lecture 1
  - Basic C++ syntax
  - Header Files
  - File Input/Output
- Lecture 2
  - Objects
    - Constructors, Destructors, New and Delete
  - Object Communication
  - Operator Overloading
  - Inheritance
- Lecture 3
  - Polymorphism and Interfaces
  - Templates and STL algorithms

C++ Programming for Physicists

Lecture Overview

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

Programming Methodology
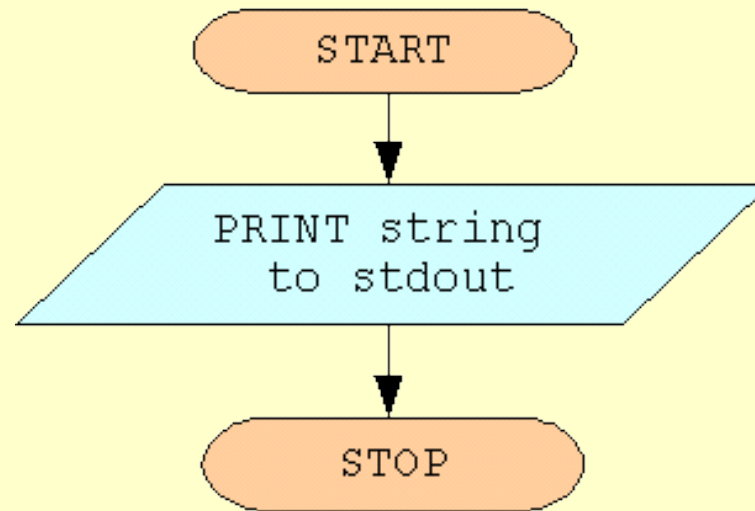
1. Requirements

   Define usecases

1. Design

   Use fundamental OO design concepts (and UML)

1. Implementation

   Implement Code (and test)

1. 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

```cpp
int main() {

  return 0;
}
```

Extract from InTheBeginning.cc

SUPA

A first C++ program

```cpp
#include <iostream>



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

extract from InTheBeginning.cc

A first C++ program

```cpp
/* 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

SUPA

Variable Declarations and Definitions

- Declaring a variable reserves some memory for it.

```cpp
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:

```cpp
int i; //declaration
i=3;    //initialisation
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

```cpp
#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

SUPA

Functions

A function is declared with

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

```cpp
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

```cpp
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).

!	Not

Reverse the result of a boolean

&&	And
||	Or

For comparing two booleans

C++ Programming for Physicists

Loops

```cpp
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

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

- iterates 1 or more times, while the condition is true.

- Can just use `while(condition){}` : iterates 0 or more times.

Conditional Statements

```cpp
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

```cpp
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.

```cpp
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];
```

| int | int | int | int |

- 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

```cpp
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++;
  }
}
```

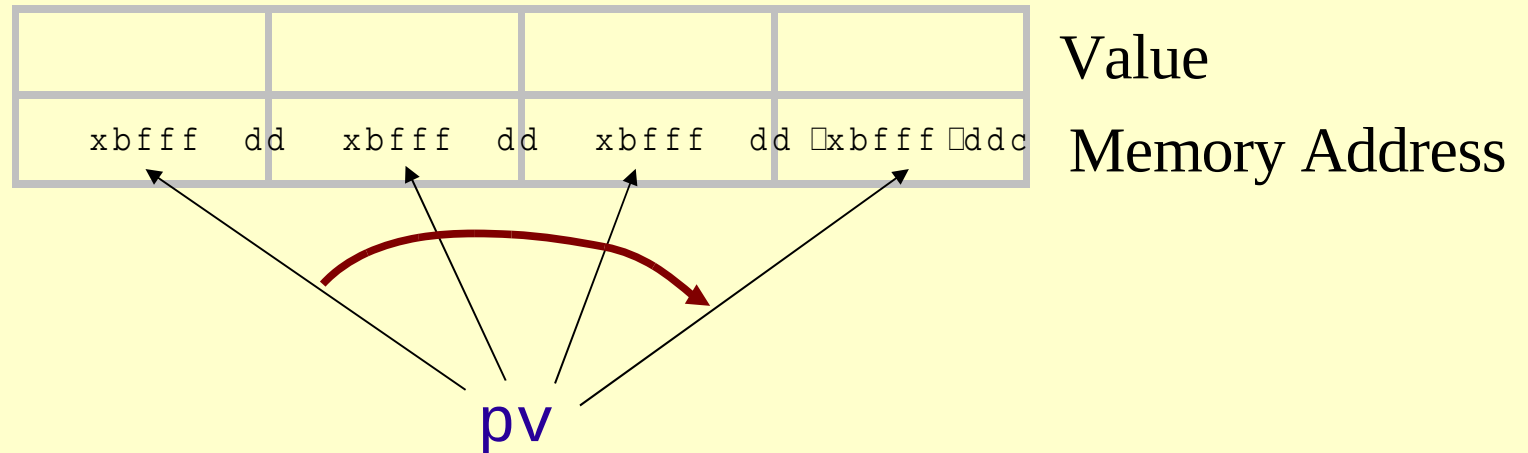Extract from ex6/Pointers.cc

```cpp
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

				Value
xbfff dd	xbfff dd	xbfff dd	xbfff ddc	Memory Address

pv

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

Pointers and Functions

```cpp
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();
  ...
}
```
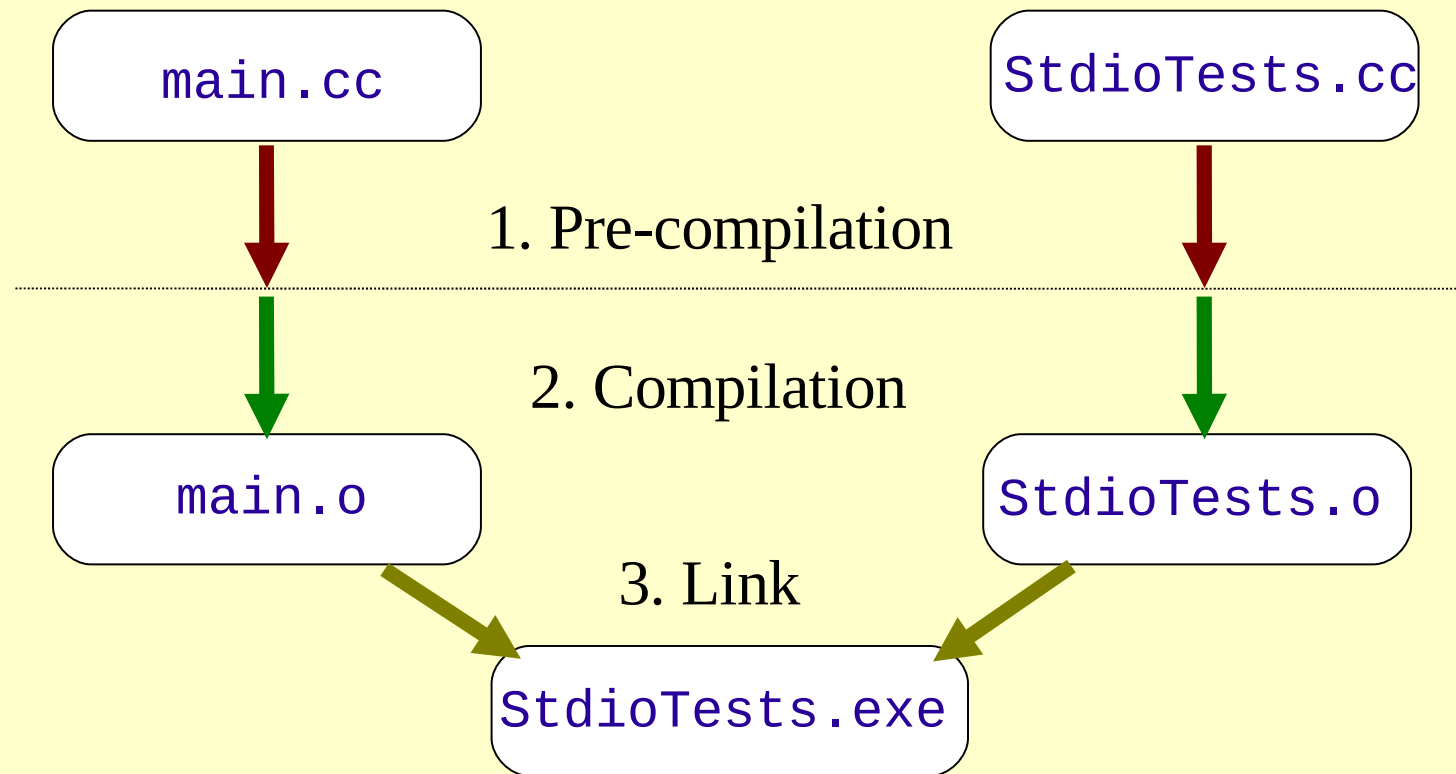
Must be in the include path

Extract from ex7/main.cc

Building an Executable

```
main.cc
```

```
StdioTests.cc
```

1. Pre-compilation

2. Compilation

```
main.o
```

```
StdioTests.o
```

3. Link

```
StdioTests.exe
```

- When linking with `g++, ld` is used

- The `ld` command line depends on which gnu compiler is used

Command Line Arguments

Number of arguments given to the command line

```cpp
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

- The C++ stream library provides an *extensible* mechanism for delivering text to and from the user.

- The stream being an abstraction that represents a device on which input and output operations are performed.

- 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

```cpp
#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

```cpp
#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

Examples and Problems

- Prepare for tutorial session (Monday 25th October)

  - Purchase recommended textbook - *Ivor Horton's Beginning C++ and The C++ Programming Language (Bjarne Stroustrup)*

  - Download session 1 examples from

  - http://my.supa.ac.uk/course/view.php?id=17

  - Build and test examples

  - Attempt problems.

- Tutorial: 10-12, linux lab, room 320, Kelvin Building, University of Glasgow.

  - http://my.supa.ac.uk/mod/wiki/view.php?id=2598