Introduction

- Why use C++?

  ANSI standard

  Compilers available on every platform

  All libraries (to my knowledge) have C and/or C++ API

- Is C++ better than Fortran?

  Structure

  Object orientation

  Reusable code and library creation

  Excellent error checking at compile time

Course layout

- Week 1

- Learn about C, the subset of C++
- (not all C covered, some C++ stuff used)

- Week 2

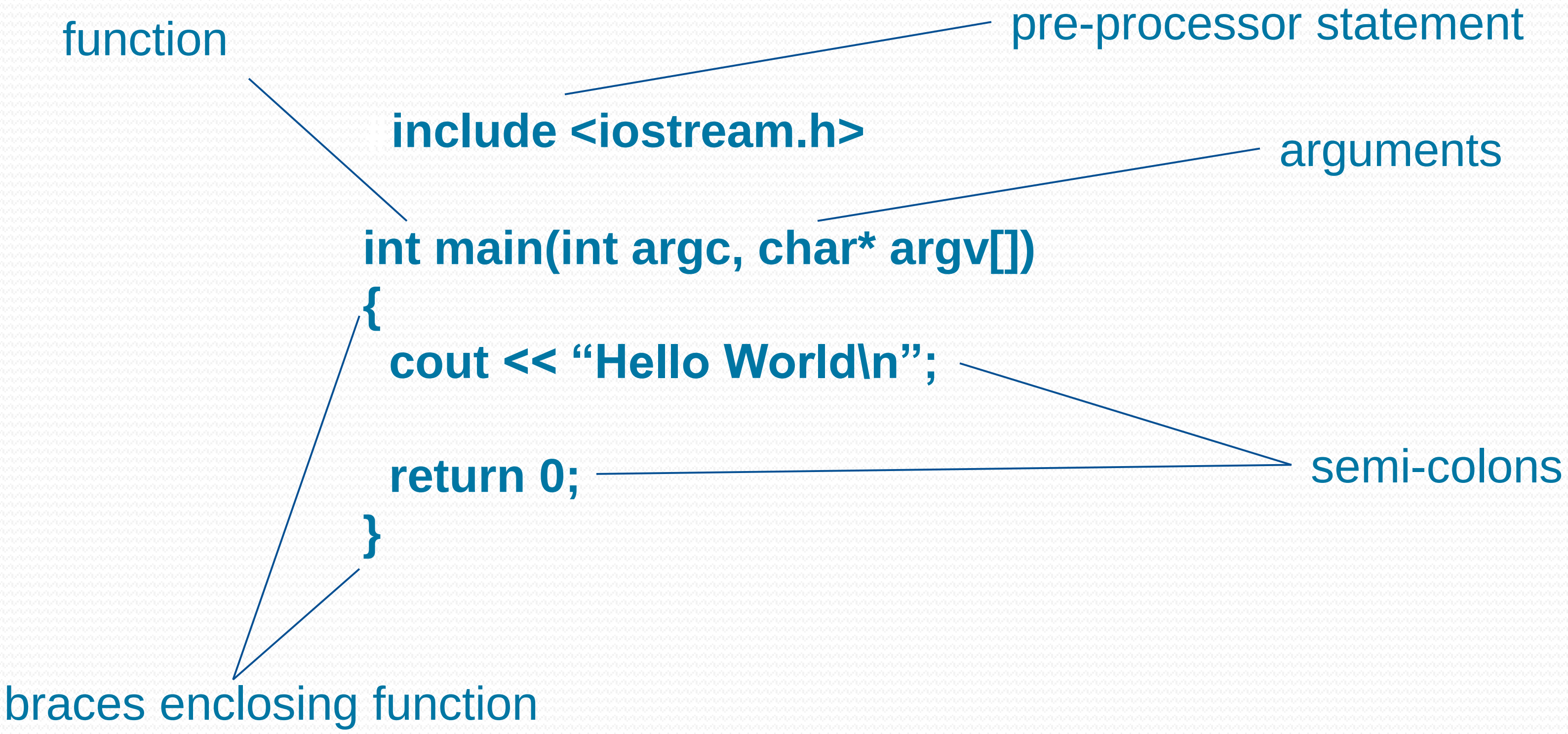
- C++ proper; classes and objects

- Quick question and answer session after each section

Sections

1) Form of a C++ program
2) Data types
3) Variables and scope
4) Operators
5) Statements
6) Arrays
7) Pointers
8) Functions
9) Structures
10) Console I/O
11) File I/O
12) Pre-processor instructions
13) Comments
14) Compiling examples under Unix

Form of a C++ program



function

pre-processor statement

```
  include <iostream.h>

int main(int argc, char* argv[])
{
  cout << “Hello World\n”;

  return 0;
}
```

arguments

semi-colons

braces enclosing function

*All C++ programs must have one, **and only one**, main function*

Some differences with Fortran

- C++ is case sensitive: x != X
- You can write it all on one line - statements separated by ‘;’
- There are NO built-in functions but lots of standard libraries
- There is NO built-in I/O but there are I/O libraries
- There are about 60 keywords - need to know about 20?
- C++ supports dynamic memory allocation
- All C++ compilers are accompanied by the C pre-processor
- You can potentially screw up the operating system with C++

Including header files and library functions

- programmer chooses what to include in the executable

Header files declare the functions to be linked in from libraries…

#include <iostream.h> ——— read & write functions

#include <math.h> ——— sqrt, log, sin, abs, *etc*

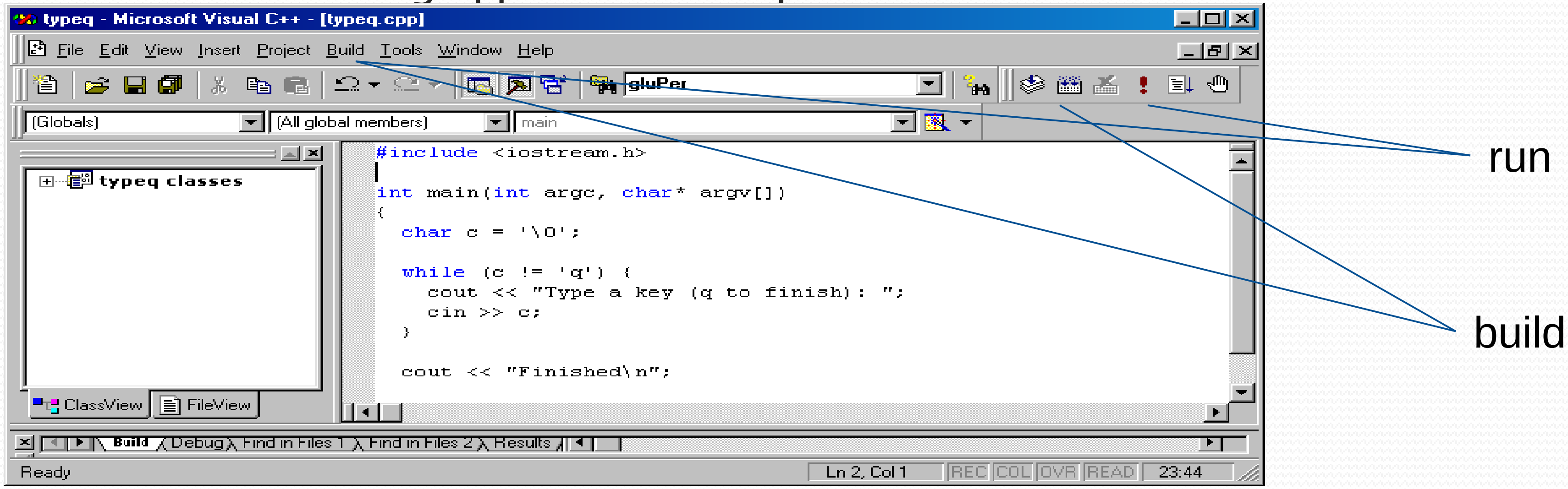
#include “D:\andy.h″ ——— My stuff that I use frequently and don’t want to recreate

If header is in “” the filename must be full (or relative);
if in <> pre-processor searches in pre-defined folders

Microsoft Visual C++

- Each program is created within a ‘project’
- A project can only contain ONE ‘main’ function
- Basically each individual project is a program
- Each project is stored in a folder
- To open a project, double-click the ‘.dsw’ file in the folder
- Click Help>Index and type a query - massive documentation
- Or select (highlight) a keyword and press F1 on keyboard

VC++ is a *big* application: a separate course!



```
#include <iostream.h>

int main(int argc, char* argv[])
{
  char c = '\0';

  while (c != 'q') {
    cout << "Type a key (q to finish): ";
    cin >> c;
  }

  cout << "Finished\n";
```

Data types

C++ is a programmer's language - need to know the basics…

- There are 6 atomic data types:

1) char - character (1 byte)
2) int - integer (usually 4 bytes)
3) float - floating point (usually 4 bytes)
4) double - double precision floating point (usually 8 bytes)
5) bool - *true* or *false* (usually 4 bytes)
6) void - explicitly says function does not return a value and can represent a pointer to any data type

Size of the data types depends on machine architecture e.g. 16 bit, 32 bit or 64 bit words

Other data types are derived from atomic types e.g. long int

Can use 'typedef' to alias your own data type names; defining C++ classes creates new types

Variables and scope

```
int a, b, c;
a = 1;
b = c = 0x3F;
```

```
float iAmAFloat = 1.234;
double iAmADouble = 1.2e34;
```

```
{
  int i, a;

  for (i=0; i<10; i++) {
    a = i;
    int b = i;
  }
  b = 2;
}
```

‘a’ declared outside loop braces

‘b’ declared inside braces; ‘b’ is in scope inside braces

‘a’ used inside braces, OK

‘b’ is unknown outside braces: ‘b’ is out of scope, ERROR

Operators

Obvious:	+, -, *, /
Shorthand:	+=, *=, -=, /=
Modulus:	%
Decrement:	--
Increment:	++
Relational:	==, !=, <, >, <=, >=
Logical:	!, &&, ||, &, |, ^, ~

int a, b;

a++;
b--;
means the same as
a = a + 1;
b = b - 1;

5%3 evaluates to 2 (the remainder of division)

Statements

A statement is a part of the program that can be executed

Statement categories:

1) Selection
2) Iteration
3) Jump
4) Expression
5) Try (exception handling; look it up)

Statements specify actions within a program. Generally they are responsible for control-flow and decision making: e.g. **if (***some condition***) {***do this***} else {***do that***}**

Selection: 'if'

General form is:

```
if (expression) {
  statement;
}
else if (expression) {
  statement;
}
.
.
.
else {
  statement;
}
```

```
bool flag;
int a, b;

if (a>0 && b>0) {
  a = 0;
  b = 0;
}
else if (flag) {
  a = -1;
}
else {
  b = -1;
}
```

'expression' is any condition that evaluates to 'true' or 'false'
'statement' could be another 'if' i.e. nesting…

A note on conditional expressions

A condition is one or more expressions that evaluate to true or false

Expressions can be linked together by logical operators

```
bool flag;
double a;

(!flag)

(a>0.0 && flag)

(a==0.0 || !flag)

(a<0.0 || (flag && a>0.0))
```

evaluates ‘true’ if **flag** is ‘false’

evaluates ‘true’ if **a** is zero or **flag** is false

Selection: ‘switch’

This statement only works with integers and chars -
it only checks for identical values (*not* less or more than).

Good for using with menu choices…

```
switch (ch) {
case ‘a’:
   doMenuOptionA();
   break;
case ‘b’:
   doMenuOptionB();
   break;
.
.
.
}
```

ch of type ‘char’
(could be ‘int’)

control jumps here
if **ch** equals ‘b’

This is a label,
and so has
a ‘:’ after it

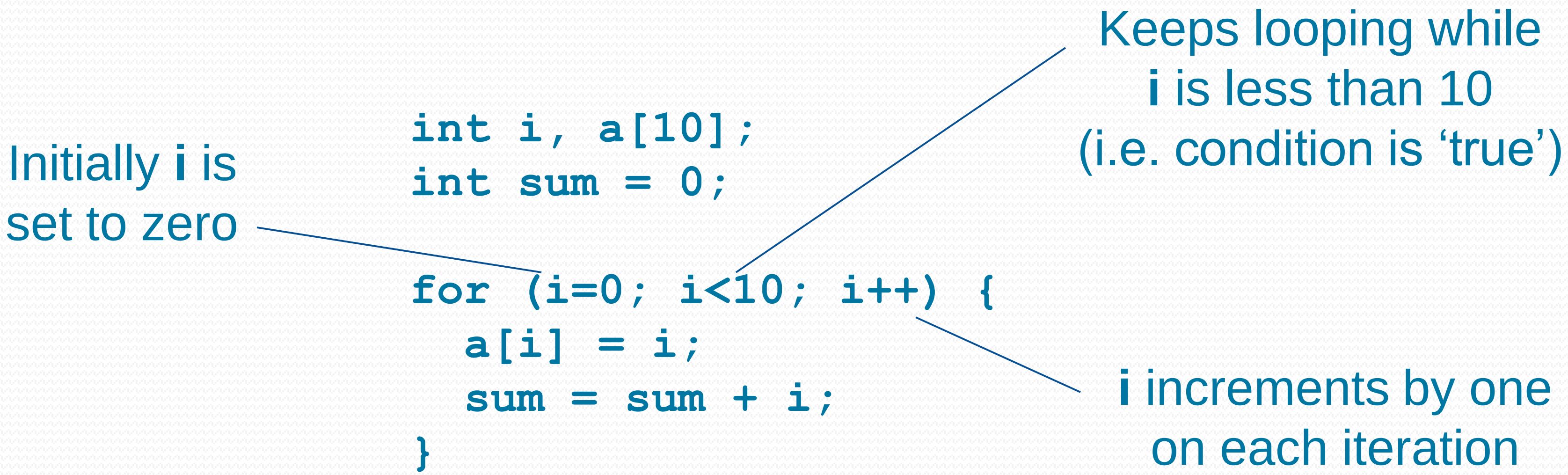
control jumps to
end brace

Note! As in ‘if’, there
is NO ‘;’ after braces

Iteration: ‘for’ loop

Huge amount of variation allowed, very flexible: generally

for (*initialisation*; *condition*; *increment*) { *statements* }



```
int i, a[10];
int sum = 0;

for (i=0; i<10; i++) {
   a[i] = i;
   sum = sum + i;
}
```

Initially **i** is set to zero

Keeps looping while **i** is less than 10 (i.e. condition is ‘true’)

i increments by one on each iteration

Iteration: ‘while’ loop

General form is

while (*condition*) { *statements* }

```
 include <iostream.h>

char c = ‘\0’;

while (c != ‘q’) {
  cout << “Type a key (q to
finish): “;
  cin >> c;
}

cout << “Finished\n”;
```

use console I/O

initialise the char variable

The loop body will execute forever - until “q” is pressed

Jump: ‘goto’, ‘return’, ‘break’ and ‘continue’

Personal bigotry: using ‘goto’ just shows bad design; do not use

‘return’ is used to exit a function, usually with a value

infinite loop

```
int a = 0;

while (true) {
  a++;
  if (a >= 10) {
    break;
  }
}
```

jump out of loop
(if loops nested, only
jump out of inner)

```
int count = 0;
int i;

for (i=0; i<10; i++) {
  if (i<3 || i>6) {
    continue;
  }
  count++;
}
```

Jump to beginning
of loop and begin
next iteration

Expressions

An expression statement is anything that ends with a ‘;’

```
func();
```

function call

```
a = b + c;

;
```

empty statement - perfectly valid

```
float root = (-b + sqrt(b*b
- 4.0*a*c)) / (2.0*a);

int flagset = 0x0001 |
0x0100;
```

bitwise OR of two integers

```
bool check = (flag &
0x0001);
```

bitwise AND of two ints; result is true or false

Arrays (1)

- In C++ arrays can be either fixed or variable length; there is a very close link between arrays and pointers

- An array can be of any data type, including your own

- C++ arrays are indexed from 0; the last element at N-1

```
int a[5];

for (i=0; i<10; i++) {
  a[i] = i;
}
```

first element at 0, last at 4

may crash when i>4; no bounds checking by default

Arrays (II)

Multi-dimensional arrays specified by:

```
double elements[100][100][100]; // about 8Mb!

for (i=0; i<100; i++) {
   for (j=0; j<100; j++) {
      for (k=0; k<100; k++) {
         elements[i][j][k] = initElement(i, j, k);
      }
   }
}
```

function that returns a ‘double’

Arrays can be initialised with contents:

float coeff[] = { 1.2, -5.456e-03, 200.0, 0.000001 };

compiler sizes array automatically

Arrays (III)

Character strings are arrays of type char
Many functions dedicated to manipulating strings <string.h>
C++ offers the standard ‘string’ class
Many library functions require you to use char* variables

A	N	D	Y		H	a

```
char myName[128];
int len;

strcpy(myName, “Andy
Heath”);
```

terminator

```
char* fullname = myName;
len = strlen(fullname);
```

length is 10

```
char* surname = &myName[5];
len = strlen(surname);
```

length is 5

```
char* argv[];
```

array of strings passed from command line

Pointers (1)

- A pointer is a variable; just like any other type of variable
- The variable contains the memory address of a certain data type
- The address is in either stack memory or heap memory
- Any atomic or user-defined data type can be pointed to
- The value of the pointer’s data type can be obtained
- Obtaining the value pointed to in memory is called dereferencing
- Two unary operators are used frequently with pointers: * and &
- Using pointers incorrectly will almost certainly crash things
- Pointers are the most useful feature of C++ (in my opinion)

Pointers (II)

C++

```
float a = 3.142;
float* pa = &a;
```

pa = 1000

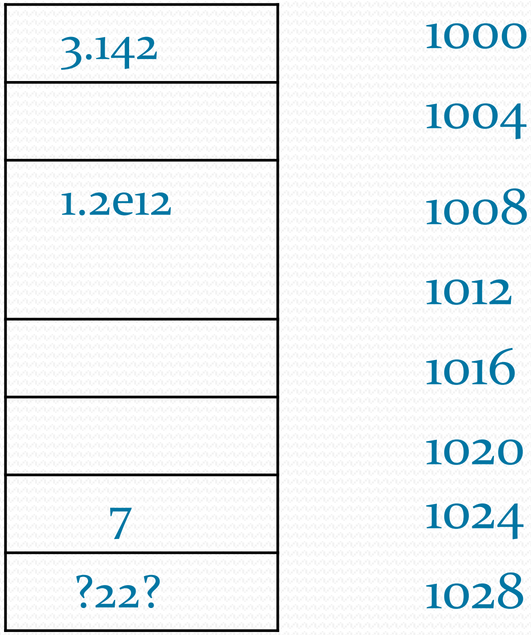
```
double b = 1.2e12;
double* pb = &b;
double bb = *pb;
```

bb = 1.2e12

```
int c = 7;
int* pc = &c;
int* pc1 = pc + 1;
*pc1 = 22;
```

potentially disastrous: something important may be at address 1028!

stack memory address



Pointers (III)

The name of an array is also a pointer:

```
float e[10], *pe;

pe = e = &e[0];
```

Can use a pointer as index into array:

```
for (int i=0; i<10; i++) {
  *pe = float(i);
  pe++;
}
```

Stack memory is allocated at compile time, when the size of an object (e.g. an array) is already known. Unless declared ‘static’, stack memory is freed when object is no longer in scope. E.g. when a function returns, the memory it’s variables occupied will be reused.

Functions (1)

A function is a distinct body of code that returns a value - even if the value is nothing at all

```
// this is some function
int func(double a, double b)
{
   if (a < b) {
      return -1;
   }
   else {
      return 1;
   }
}

// this is the main function
int main(int argc, char* argv[])
{
   return func(1.0, 2.0);
}
```

Functions (II)

C++

This is the same function - the code is written in a different order

```
int func(double, double);

// this is the main function
int main(int argc, char* argv[])
{
  return func(1.0, 2.0);
}

// this is some function
int func(double a, double b)
{
  if (a < b) {
    return -1;
  }
  else {
    return 1;
  }
}
```

Prototype
(this is what is
in header files)

Functions (III)

- A function returns a value (except if it is of type ‘void’)
- The value is specified with the ‘return’ keyword
- The value returned must be the same type as the function

```
double logit(double z)
{
  return log(z);
}
```

```
int addInts(int a, int b,
int c)
{
  return a + b + c;
}
```

A function can call itself - recursion

```
int factorial(int n)
{
  if (n > 0) {
    return n*factorial(n-1);
  }
  else {
    return 1;
  }
}
```

Functions (IV)

Arguments passed to functions in one of three ways:

1) By value (default)
2) By reference
3) Thru a pointer to the argument

```
  float x = 2.0f;

  func(x);

  cout << x;
}

void func(float x)
{
  x  = pow(10, x);
}
```

C++ default

```
  float x  = 2.0f;

  func(x);

  cout << x;
}

void func(float& x)
{
  x  = pow(10, x);
}
```

Like Fortran

```
  float x = 2.0f;

  func(&x);

  cout << x;
}

void func(float* px)
{
  *px = pow(10, *px);
}
```

Thru a pointer

Structures

- Structures are a C concept
- Replaced in C++ with the ‘class’ concept
- Objects of a class can be completely defined as a new data type

```
class Andy {
  // definition of stuff
};

int main(int argc, char*
argv[])
{
  Andy a, b, c;

  a = b + c;
  cout << a;
}
```

a, b and **c** are of type **Andy**

If we choose to, we can give ‘Andy’ objects the ability to do addition, and to write themselves to the console

but that’s for NEXT WEEK…

Console I/O

- The console is a command window (DOS or terminal)
- C++ uses the ‘<<’ and ‘>>’ operators to write and read
- C++ recognises what sort of data you want to read or write and acts accordingly
- Use “manipulators” to format I/O - look it up!

```
  include <iostream.h>

int main(int argc, char* argv[])
{
   char* name = “Andy Heath”;
   float number = 1.2345;
   int menuChoice;

   cin >> menuChoice;

   cout << name << “ “ << number <<
“\n”;

   return 0;
}
```

read an ‘int’ token from the keyboard - tokens separated by white-space (space, newline, tab)

write variables and constants to screen; use newline (‘\n’) to finish line

File I/O

- oSimilar to console I/O
- o Different types of stream for input and output
- o File streams defined in <fstream.h>

```
ifstream inFile;
ofstream outFile;
float number;

inFile.open(“readme.txt”);
outFile.open(“writeme.txt”);

while (!inFile.eof()) {
  inFile >> number;
  outFile << number;
}

inFile.close();
outFile.close();
```

Keep going until the end of file is reached

These are *methods* of the base classes ‘ios’ and ‘fstream’… ‘inheritance’ covered next week!

Pre-processor instructions

Pre-processor instructions allow the programmer to choose what source code the compiler sees - very useful for writing programs that are portable between Unix and Windows

```
ifdef WIN32
#include <someWindowsHeader.h>
void myFunc(/* Windows data
types */);
#else
#include <someUnixHeader.h>
void myFunc(/* Unix data types
*/);
#endif
```

Can use verbose output in a function to help with debugging

```
#ifdef _DEBUG
  cout << “Debug: at end of ‘func1’ x = “ << x << “\n”;
#endif
```

Look at Project>Settings and the “C/C++” tab for Pre-processer defs

Comments

Two types of comments:

/* blah blah blah */

// blah blah blah

```
/*
 * this function computes the square root of infinity
 */
double sqrtInf()
{
  double infinity = NaN; // is infinity a number?

  return 1.0; // near enough since I haven’t got a clue
}
```

VC++ colours comments green by default