C++ Crash Course

for physicists

Morning 3h: Basics

45 mins slides + 2h15 hands-on

START

Afternoon 3h: Advanced

15 mins slides + 2h45 hands-on

START

P. Skands - Monash University - Feb 2015

Content

Basics

- Compiled Code
- The **main** program
- The Standard Library (STL)
- Scope
- Loops
- Functions, Modularity, Libraries
- Make & Makefiles
- Vectors and Maps

Advanced

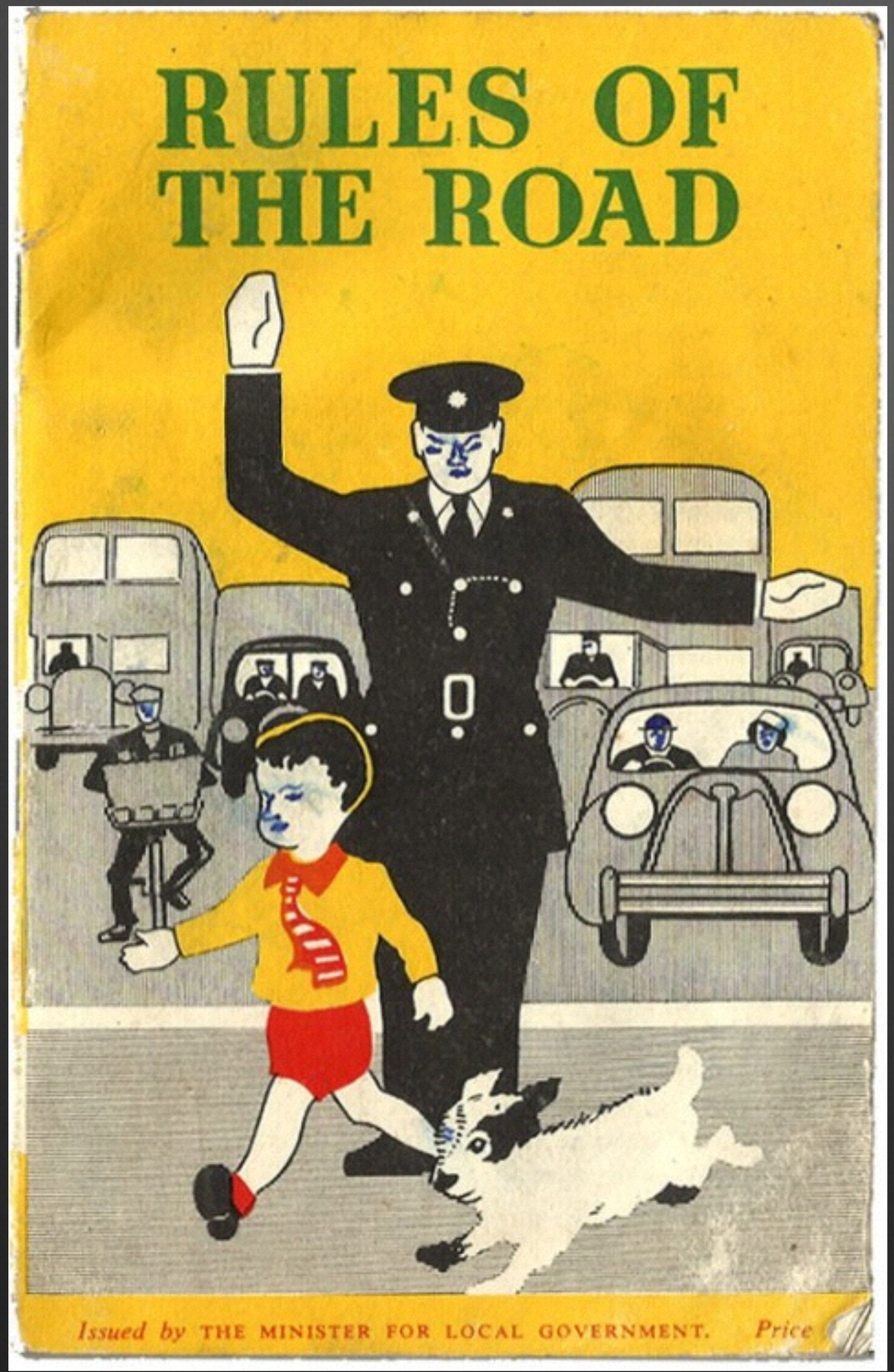
- Pointers (& memory)
- Classes
- Working with a real code

Beware:

Inheritance, templates, iterators, inlining, operator overloading, shared libraries, preprocessor directives, compiler flags, exception handling, and *much else* not covered here.

Disclaimer

- This course is ultra brief
- Focus on concepts
- Aim: get to be able to write and work with some code



Disclaimer

Still, this is pretty dry stuff

At the end of today, use it to collide particles

Compiled Code

Same principle as FORTRAN

Code

模式	指示符
ECI	0111
数字	0001
字母数字	0010
8 位字节	0100
日本汉字	1000
中国汉字	1101
结构链接	0011
	0101 （第一位置）

Binary
(machine code)

```
int main() {
  // This is an example code
   int someNumber = 4;
   int otherNumber = 5;
   int sum = someNumber + otherNumber;
   // Exit program. Return status code
   return 0;
}
```

```
00000000: cffa edfe 0700 0001 0300 0080 0200 0000  ................
00000010: 1000 0000 6803 0000 8500 2000 0000 0000  ....h..... .....
00000020: 1900 0000 4800 0000 5f5f 5041 4745 5a45  ....H...__PAGEZE
00000030: 524f 0000 0000 0000 0000 0000 0000 0000  RO..............
00000040: 0000 0000 0100 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 1900 0000 3801 0000  ............8...
00000070: 5f5f 5445 5854 0000 0000 0000 0000 0000  __TEXT..........
00000080: 0000 0000 0100 0000 0010 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0010 0000 0000 0000  ................
000000a0: 0700 0000 0500 0000 0300 0000 0000 0000  ................
000000b0: 5f5f 7465 7874 0000 0000 0000 0000 0000  __text..........
000000c0: 5f5f 5445 5854 0000 0000 0000 0000 0000  __TEXT..........
…
```

Source Code

main.cc

Command

```
g++ main.cc
```

Executable

a.out

(assuming your C++ compiler is called g++)

Think a.out is a stupid name?

The computer doesn't care

Source Code	Command	Executable
main.cc	g++ main.cc	a.out
main.cc	g++ main.cc -o main.exe	main.exe
main.cc	g++ main.cc -o main	main

To compile and execute code:

Nothing happens, because we are not writing anything to the screen yet.

```
> g++ main.cc -o main
> ./main
>
```

The Standard Library

http://www.cplusplus.com/reference/

Your default toolbox

- To get some output, we'll use some functionality from the "standard library", a very useful box of tools to start from.

Just an example. Lots more where that came from. Google it.

```
// STL headers are put in <…> brackets.
// Include the STL header file that deals with input- and output streams
#include <iostream>

// Having included that header file, we can now use it in our main program

int main() {
  // This is an example code
   int someNumber = 4;
   int otherNumber = 5;
   int sum = someNumber + otherNumber;
   // Write out result to the screen
   std::cout << sum << std::endl;
   // Exit program. Return status code
   return 0;
}
```

You'll see many of these include statements in real C++ code

Ordinary code lines always end on ";"

"std::" means: look for these functions in the **namespace** "std"

(see next slide)

The **std::** namespace and **using std**

Namespaces

Disambiguation

- When you link lots of code together, what if several different variables have the same name? **Namespaces** protect against that.

  *E.g., stuff from the Standard Library lives in the namespace **std***

- Since we use the std functions a lot, let's **include** that namespace

```
// Include the STL header file that deals with input- and output streams
#include <iostream>

// Automatically look for things in the std namespace
using namespace std;

int main() {
   int someNumber = 4;
   int otherNumber = 5;
   int sum = someNumber + otherNumber;
   // Write out result to the screen
   cout << sum << endl;
   // Exit program. Return status code
   return 0;
}
```

The code has gotten easier to read, more compressed, at the price of being less explicit about where "cout" and "endl" are really coming from.

The "using" statement means we now automatically look in the std namespace

Scope

with if … then … else example

- In C++, variables are automatically created and destroyed

  (This saves memory, compared with never killing them, but it means you have to think about what's alive and what's dead)

```
// STL headers and namespace
#include <iostream>
using namespace std;

int main() {
   int someNumber  = 4;
   int otherNumber = 5;
   int sum = someNumber + otherNumber;
   if (sum != 9) {
      string message="You cannot count";
      sum = 9;
   } else {
      string message="You count just fine";
   }
   // Print whether things went well or not
   cout<<message<<endl;
   // Exit main program
   return 0;
}
```

This isn't going to work.

The variable "`message`" only exists inside each of the `if` clauses separately. Destroyed when they end.

I.e., it does not exist outside those "scopes".

(But since "`sum`" exists globally, the part where it is reset to `9` does work)

Scope

with if … then … else example

- In C++, variables are automatically created and destroyed

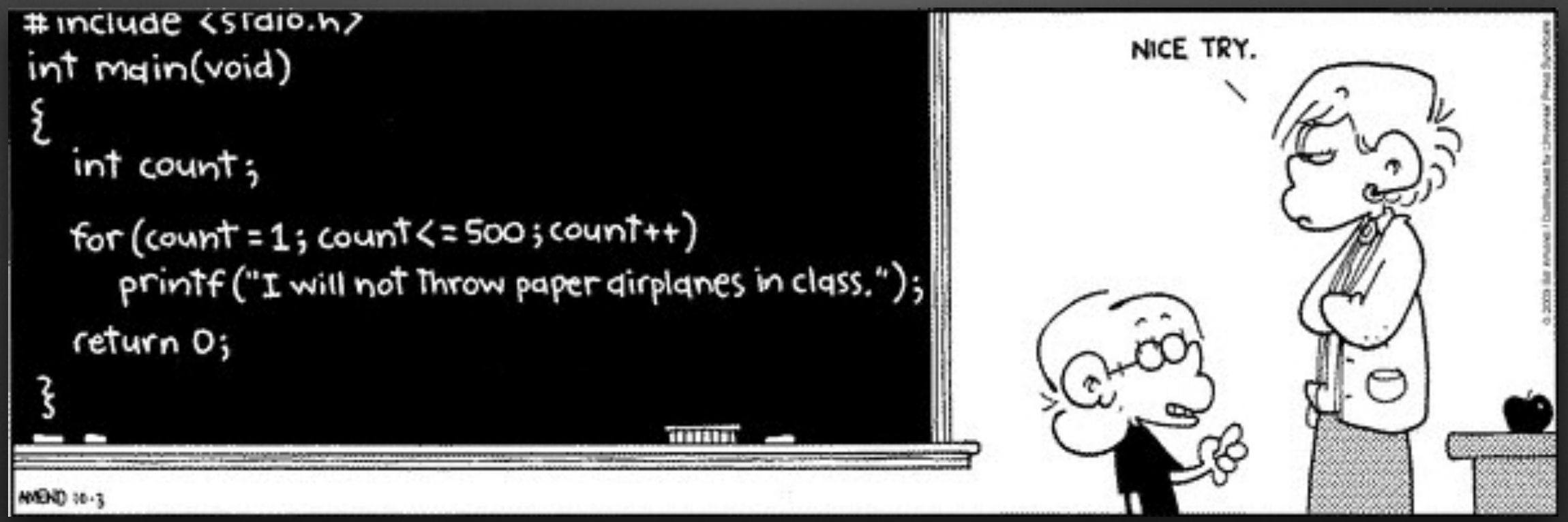
  (This saves memory, compared with never killing them, but it means you have to think about what's alive and what's dead)

```
// STL headers and namespace
#include <iostream>
using namespace std;

int main() {
   int someNumber  = 4;
   int otherNumber = 5;
   int sum = someNumber + otherNumber;
   string message;
   if (sum != 9) {
      message="You cannot count";
      sum = 9;
   } else {
      message="You count just fine";
   }
   cout<<message<<endl;
   // Exit main program
   return 0;
}
```

Solution:

Move declaration of `message` outside the `if ()` scope.

Loops



- `printf("…")` is old-fashioned C. In C++, use `cout<<" … "<<endl;`
- `count++` : increase the variable `count` by one *(hence the name C++)*

```
// Pseudocode for a "for" loop.
for (starting condition; ending condition; iteration operation) {
 …
}
```

For and While

and ++i vs i++

```
// Pseudocode for a “for” loop.
for (int i=1; i<=500; i++) {
   cout<<“I will not throw paper airplanes in class”<<endl;
}
```

```
// Pseudocode for a “while” loop.
int i = 0;
while (++i <= 500) {
   cout<<“I will not throw paper airplanes in class”<<endl;
}
```

```
// Alternative pseudocode for a “while” loop.
int i = 0;
while (i++<=500) {
   cout<<“I will not throw paper airplanes in class”<<endl;
}
```

```
++i <= 500 : add 1, then compare (preferred today)
i++ <= 500 : compare using original i, then add 1
```

Some nice tricks:

```
i += 5; // Add 5 to i
i *= 2; // Multiply i by 2
i /= 2; // Divide i by 2 (but beware integer division! E.g., 5/6 = 0, but 5.0/6.0 = 0.8333)
```

Also works with strings (example of overloading)

```
message += “ appended text”;
```

Functions

- If you know you're going to be using the geometric mean of two integers a lot, encapsulate it in a function

  Note: sqrt() resides in the cmath header, so we must include that too

```
// STL headers and namespace
#include <cmath>
#include <iostream>
using namespace std;

// You can put functions above your main program
double geoMean(int i1, int i2) {
   return sqrt(i1*i2);
}

int main() {
   int someNumber  = 4;
   int otherNumber = 5;
   double mean = geoMean(someNumber,otherNumber);
   cout<<"Geometric mean is = "<<mean<<endl;
   // Exit main program
   return 0;
}
```

Note: this function will happily take negative inputs and will then happily crash. Protecting against garbage parameters is important but not part of this tutorial

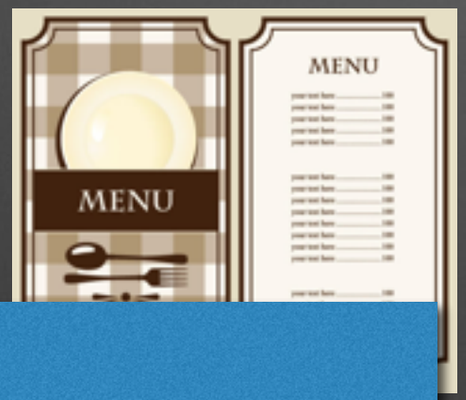
Note also: only takes integer inputs. Kind of special purpose. Better to define in terms of doubles.

Modularity

Someone asked you to produce a code to calculate the geometric mean.
How would you deliver it? As a **library** which they can **link** to.

Header File

geomean.h




```
// Headers and namespace
#include <cmath>
using namespace std;

// Avoid name clashes: define a namespace
namespace averages {
// List of functions provided
  double geoMean(int i1, int i2);
}
```

+ Source Code

geomean.cc

```
// Put all declarations in .h file.
#include “geomean.h”

// The .cc file contains the meat
double averages::geoMean(int i1, int i2) {
   return sqrt(i1*i2);
}
```

Command

```
g++ -c geomean.cc
```

Object File

geomean.o

(machine code)

Contains the compiled code for this code piece

Linking

Same principle as FORTRAN

So you got your geomean code compiled.
How do you use it?

main.cc

```
// Include headers and namespace
#include <iostream>
include “geomean.h”
using namespace std;
using namespace averages;

int main() {
   int someNumber  = 4;
   int otherNumber = 5;
   double mean = geoMean(someNumber,otherNumber);
   cout<<“Geometric mean is = “<<mean<<endl;
   // Exit main program
   return 0;
}
```

Note: at the time main.cc is compiled, it needs to have access to the header file geomean.h. That means I need to have a copy of it, in addition to geomean.o, and I need to know where both of those files reside.

Command

```
g++ main.cc geomean.o -o main
```

Executable

main

(machine code)

Libraries

More precisely, “static libraries”; shared ones not covered here

Same principle as FORTRAN

- Libraries are collections of object files:
  - You can create one, `libgeomean.a`, by using the “ar” utility, which should exist on your unix system

```
ar cru libgeomean.a geomean.o stuff.o otherstuff.o
```

- You can link your main program to them

Command

```
g++ main.cc -o main libgeomean.a
```

Executable

`main`

(machine code)

Often, you will link your code to several libraries, and they won’t all be in the same place.

```
g++ main.cc -o main -I/usr/local/include -L/usr/local/lib -lgeomean -larithmean
```

-I: include path for header files

-L: include path for library files

-l: shorthand

Make & Makefiles

Same principle as FORTRAN

- Say you've got a couple of auxiliary .cc files. You want to compile them into objects, put them in a library, and link your main program to it

Makefile

```
# Define what target we normally want to make
default : main

# Define a variable. This one a list of objects to include in libgeomean.a
LIBOBJECTS = geomean.o

# This defines the rule for creating libgeomean.a
libgeomean.a : $(LIBOBJECTS)
	ar cru libgeomean.a $(LIBOBJECTS)

# This defines the rule for creating geomean.o from geomean.cc and geomean.h
geomean.o : geomean.cc geomean.h
	g++ -c geomean.cc

# Make the main program
main : main.cc libgeomean.a
	g++ main.cc -o main libgeomean.a

# Normally we also define a way to clean up
clean :
	rm -f main ./*.o ./*.a
```

note: use tabs

```
> make
g++ -c geomean.cc
ar cru libgeomean.a geomean.o
g++ main.cc -o main libgeomean.a
>
```

C++ Vectors

http://www.cplusplus.com/reference/vector/vector/

- Vectors are examples of a C++ ***container***
- Data types designed to store other data

```
// Include headers and namespace
#include <vector>
using namespace std;

int main() {
   vector<int> numbers;
   // Put some numbers on the “back” of the vector
   numbers.push_back(4);
   numbers.push_back(5);
   double sum = numbers[0] + numbers[1];
   // Exit main program
   return 0;
}
```

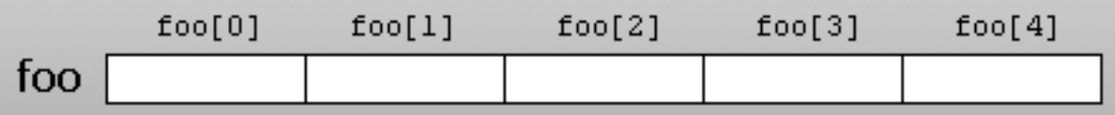
For simple tasks, you can also use an *array*

```
int numbers[2];
numbers[0] = 4;
numbers[1] = 5;

or:
int numbers[2] = {4, 5} ;
```

```
// Alternative with a loop. Start sum off at zero.
double sum = 0.0;
// Determine length of vector (= length of loop)
int length = numbers.size();
for (int i=0; i<=length; ++i) sum += numbers[i];
```

Wrong. Should be < not <= Why?

C/C++ start counting at zero



C++ Maps

http://www.cplusplus.com/reference/map/map

- Maps are examples of a C++ ***container***
- Data types designed to store other data

```
// Include headers and namespace
#include <iostream>
#include <map>
using namespace std;

int main() {
   map<string,double> salaries;
   // Put some salaries in the map
   salaries[“Alice”]=200000.0;
   salaries[“Bob”]  =150000.0;
   // Print out the salaries
   cout<<“The salary of Alice is $”<<salaries[“Alice”]<<endl;
   cout<<“The salary of Bob is   $”<<salaries[“Bob”]<<endl;
   // Exit main program
   return 0;
}
```

Note: looping over map entries requires the use of **iterators** (intuitively, you *iterate* through the entries, since they are not numbered). Not included here. *If you need them, google them.*

You now know a few basics

Time to take a test drive

Problems

- Using what you have learnt in these slides, write a simple main program that writes “hello world” in the terminal.

- Using loops, compute and write out the first 10 terms of the Fibonacci sequence; 0, 1, 1, 2 , 3, 5, 8, 13, … ; then try 50 Fibonacci numbers.

- Encapsulate your Fibonacci calculator as a function, and call it from your main program. The writing out of the numbers should still be done in the main program.

  Recursively? Consider efficiency and speed. The Unix “time” command can be used to check execution speed. E.g.: `time ./a.out`

- Put your Fibonacci calculator in a namespace, to disambiguate it.

- Split the Fibonacci calculator off as a separate c++ “library”, fibonacci.cc and fibonacci.h. Include them in your main program, and link to the library.

- Write a Makefile to handle the dirty work.

Advanced C++

... in 3 hours

Some kind advice: Failing to understand pointers when writing C code is just waiting to shoot yourself in the foot, if not the head.

Memory

- In C++ you can ask what the memory location of anything is. Let's try:

```
#include <iostream>
using namespace std;

int main ()
{
   int  var1;
   double var2;

   cout << "Address of var1 variable: " << &var1 << endl;
   cout << "Address of var2 variable: " << &var2 << endl;

   return 0;
}
```

The & (address-of) operator tells us where the variable is located in memory

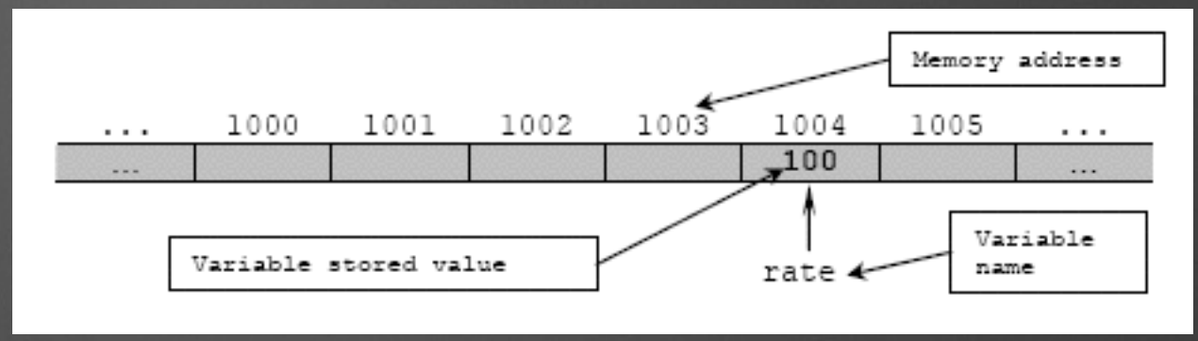
Pointers

- We can refer to a variable by using its location in memory (so long as that location doesn't change).
- A *pointer* contains such a memory location, together with information on how to interpret the data found there (is it int, double, or whatever…)

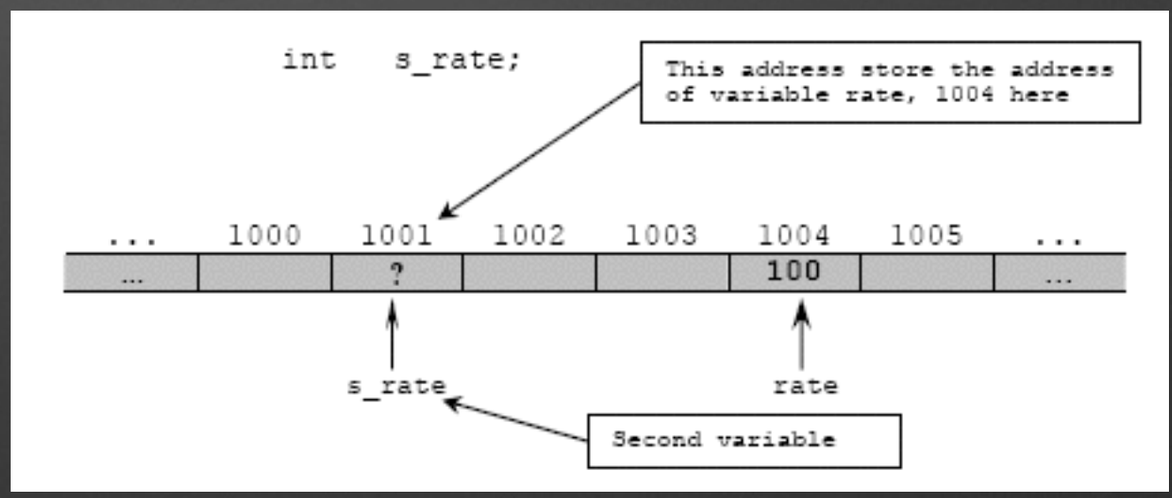
```
#include <iostream>
using namespace std;

int main ()
{
   // Declare a normal integer, then declare a pointer to an int
   int  var1 = 10;
   int  *intPtr;
   // Let the intPtr point to the location of var1
   intPtr = &var1;
   cout<<"The address of var1 is "<<intPtr<<endl;
   // Since intPtr knows it is a pointer to an int,
   // we can dereference it to find out what's actually there.
   cout << "The value at that address is " << *intPtr << endl;
   return 0;
}
```

Pointers and Memory

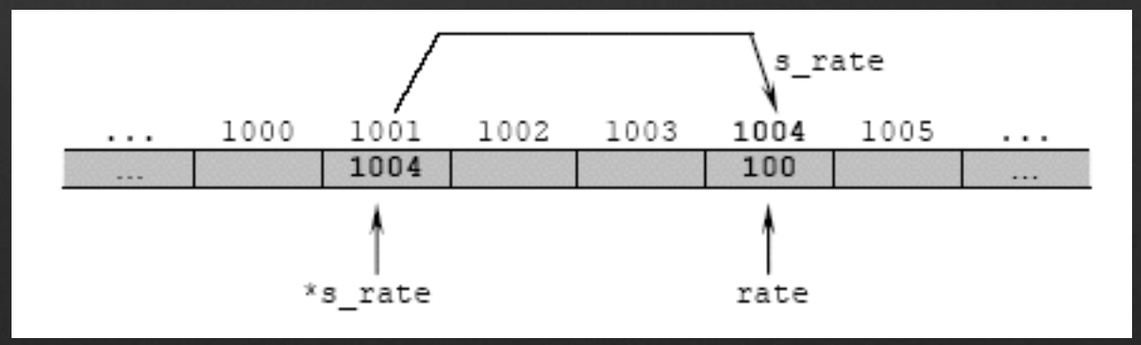
int rate = 100;



int *s_rate;
(value not specified yet)



s_rate = &rate;



Note: you can even create a pointer to a new object in one go, using *new,* not covered here.

Some major uses of Pointers

- 1) You have lots of some kind of variable. You'd like to do a loop where each one successively is used and/or modified. You can collect them into a vector, ***or*** you can create a pointer to such a variable and let that point to each one in succession, and then do the operations using the pointer.

  Imagine you a very complicated data structure. You wouldn't necessarily want to go to the trouble of creating a vector of such objects, which would slow you down as well as increase your memory usage.

- 2) Large program with complicated data structures. Define one *instance* of each structure. Everyone else gets passed a pointer to that instance.

  Otherwise you risk ending with a proliferation of objects burning memory and being out of sync with each other.

- 3) Sometimes it's just easier to say the real one lives over **there**

- 4) Memory management (again mainly for large complex programs)

Caution: things can move in memory. Reallocations.

Values and References

- When you call a function in C++, a new copy of that variable is created in the function you called. The original remains unmodified. Only the value is passed, not the variable itself.

```
// This function doesn’t do anything
void timesTwo(int i1) {
   i1 *= 2;
}
// i1 is modified locally inside this function, but
// the calling function doesn’t know or care.
```

- So if you actually want to give the function your variable to modify?

```
// Send the function a reference.
void timesTwo(int& i1ref) {
   i1ref *= 2;
}
// This function does modify the original variable
// The reference is essentially a memory address,
// like a pointer, but without the need to dereference
```

Classes

- Classes are generalised containers which can contain not only data but also functions (called methods)

rectangle.h

```
// Header: example of a class
class Rectangle {
    int width, height;
  public:
    void setDimensions(int,int);
    int area() {return width*height;}
};
```

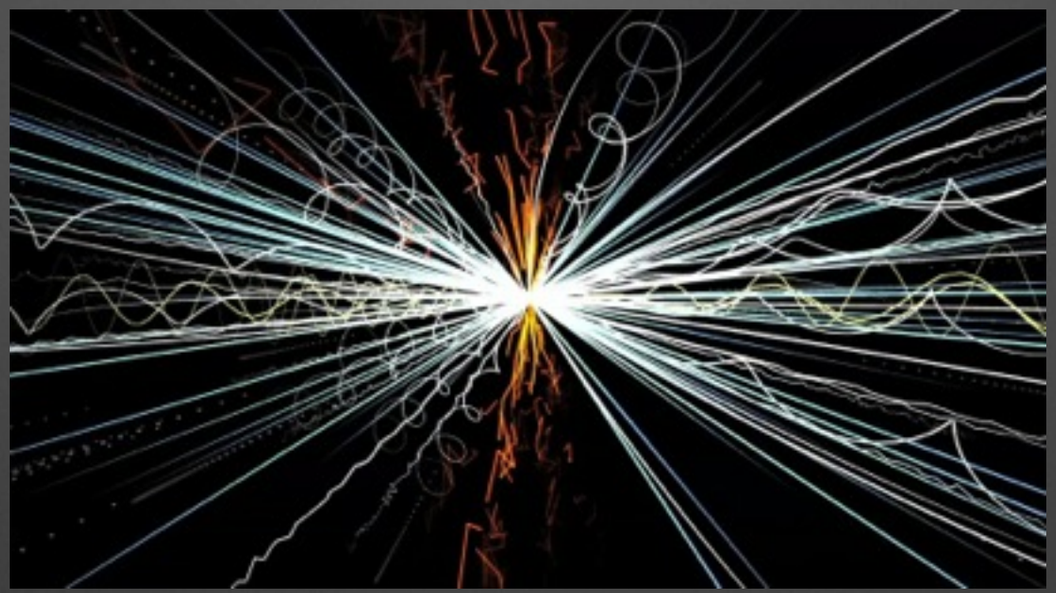
rectangle.cc

```
// Implementation
#include “rectangle.h”
void Rectangle::setDimensions (int x, int y) {
  width = x;
  height = y;
}
```

program.cc

```
// Main program
#include <iostream>
#include “rectangle.h”
using namespace std;
int main () {
  Rectangle rect;
  rect.setDimensions(3,4);
  cout << "area: " << rect.area() << endl;
  return 0;
}
```

Working with Real Code



- We will now use a state-of-the-art C++ code to simulate particle collisions at the Large Hadron Collider
- Instructions : PDF
- PYTHIA Homepage