Some Notes on C++

Nick Maclaren

nmm1@cam.ac.uk

November 2015

Purpose of Lecture

Bjarne Stroustrup wrote a very useful book
Programming – Principles and Practice Using C++
I taught a course using it – needing 200+ hours' work

This lecture is some points and additions I made
Important practical issues that are rarely mentioned

For details, refer to my version of the course
C++
Some of them are also in his original:
http://www.stroustrup.com/Programming/

Topics

- Rules for using I/O safely and portably
- Advice on library use, including the STL
- Numerics, random numbers and matrices

Safe Open Modes

in	for reading only
out|trunc	new data for writing only
out|app	extend file at end
in|out	if file exists (start with a read)
in|out|ate	if file exists (start with a write)
in|out|trunc	new or overwrite (start with a write)

ifstream includes in and ofstream includes out
All can be used with or without binary

Above is only safe use of app
Never reposition if opened for app

Repositioning

Can seek only on ordinary disk files

seek(0)	to reread from start
seek(0,trunc)	to rewrite from start
seek(0,ate)	to extend at the end

- Must separate reads and writes by seek

Can seek by byte count only if all of:

- Ordinary disk file on Unix–derived system
- Opened with binary

Non-Trivial Files

- Some files are not just arrays of bytes
- Some can be opened only once – be warned!

Sockets, TTYs, etc., and non–Unix systems

- Simplex stream I/O is only reliable form for them

I.e. input–only or output–only, no repositioning

Remote files (NFS etc.) have restrictions

- If accessed in parallel, open for input–only

Use a library like HDF if you need update

I/O Errors

C++ sets a flag bit and ignores further calls
Never use clear() on the bad() bit
You can set up a stream to throw on bad errors
Slide 21 of mine (19 of his) 10_iostreams.odp

System errors on output are rarely detected
System errors on input often look like end of file

- This area is completely broken in modern systems

C is worse and continues regardless
No separation of recoverable and catastrophic
Latter leads to undefined behaviour and chaos

Formatted I/O

Many people dislike C++'s facilities, for good reason
C's are easier to use, more flexible and unsafe
There are several alternative approaches in

C++/...
.../11a_other_io.odp

Libraries, Software Reuse etc.

There is a great deal on this area in

C++/...
.../21a_Lib_issues.odp

C++/...
.../24a_more_numerics.odp

I am going to mention only a few points here
Mainly ones that are relevant to other MPhil courses

The STL (1)

It says why I don't like the STL's design much
It also describes a lot of 'gotchas' to avoid
And approaches that I regard as cleaner and simpler

Vastly the most useful are <vector> and <list>
Followed by <array>, <map> and <set>
Don't bother with <valarray>, <stack> etc.
<algorithms> isn't useful, either – code them yourself

Generally, use <vector> unless need a fixed size
Then must use <array> – cleaner than built–in arrays

The STL (2)

Watch out for shared–memory parallelism
Separate container objects are independent
Information methods are read–only on container

Separate elements are independent if left in place
Element assignment may update the whole container
Rules for iterators are full of serious ‘gotchas‘
The data are not contiguous (i.e. like C)

Exceptions: <vector>, <deque>, <array>, <string>
Replace elements, but not append, insert or erase
Can create C pointer to data, and pass to MPI etc.

Pure Data Classes (1)

Critical when passing to MPI, binary I/O etc.
Slightly stronger than a standard–layout class
Class layout is, in general, a can of worms

In simple terms, pure data classes must not contain:

- Any reference or pointer
- Any container except <array>
- Any class except a pure data class
- Any virtual functions

- And arcane restrictions on derived classes

I suggest you avoid assuming anything about those

Pure Data Classes (2)

The alignment and padding may vary considerably
Hardware, system, compiler and compiler options

- Check carefully when reading in binary files

Be very careful when using any library class
Whether C++ or external library (e.g. Boost)

- Their exact properties are very rarely defined

E.g. <complex>, <tuple> and <bitset> are pure data
<exception> is definitely not – and <mutex>?

- Almost none of this is actually specified

Numeric Error Handling

C++ 2011 changed its base from C90 to C99
C required errno for math.h – C99 broke that
Its IEEE 754 handling is solid with 'gotchas'

C++ 2011 included library calls but not pragmas
So using IEEE 754 is necessarily undefined!

- Compilers, libraries and options will differ

Ignore whole hopeless mess, and check yourself

Precision and Accuracy

Look at exercises 1a, 1b and 1c in:

C++/...

.../15_graphing.odp

They show how to solve some common problems

More, including Kahan summation etc., in:

C++/...

.../Exercises/Chapter_24

My high–precision accumulator code is online
You are welcome to use it (e.g. on GPUs)

Random Numbers (1)

Almost all of the Web and most books are erroneous

Don't use rand() in serious code – it's ghastly
Numerical Recipes and Boost::random are unreliable
C++ 2011 supersedes latter, anyway

- Only the Ranlux and Mersenne ones are any good

Knuth_b is tolerable for occasional use

Marsaglia's generators are variable in quality
I have a good generator that people are welcome to

Random Numbers (2)

Recheck important results with different generators
Interactions with program can cause spurious effects
Use ones based on different principles for safety

Parallelism is a major problem – ask me for advice
Thread quasi–independence is a very tricky problem

If initialising separately per thread/process, must

- use a very high–quality generator
- use a very long–period generator
- use randomised initial seeds

Matrices (1)

Many scientific libraries have suitable matrix classes
I tried using the STL and Boost – ugh
Much easier to write your own, as described in

C++/...

.../24a_more_numerics.odp

Exercises 15–18 help you to learn how

- Fortran storage order can be faster

Due to the use of right solution of equations
Often gains by storing matrix and matrix^T
Warning: writing an efficient transpose needs care

Matrices (2)

There is example code (both Bjarne's and mine) in

C++/...

.../Exercises/Chapter_24

You are welcome to use them, but please give credit

Algol 68 and Fortran handle subsections properly
I.e. can pass to a function as a normal array
You must use (LWB,size,stride) for each dimension
The example code above does not do that