MA400: Financial Mathematics

Introductory Course

Lecture 7: A tour of the C++ libraries

<cstdlib>

- Program termination: the `exit()` function, and the constants `EXIT_SUCCESS` and `EXIT_FAILURE`.
- The `rand()`, which returns a pseudo-random integer between 0 and the constant `RAND_MAX`.

```
// The following generated variables lie between ...
int a = rand();           // ... 0 and RAND_MAX
int b = rand() % 100;     // ... 0 and 99
int c = rand() % 100 +1; // ... 1 and 100
```

Numerical libraries

- `<cmath>`
  The common mathematical functions are defined here, e.g. `cos()`, `sin()`, `tan()`, `exp()`, `log()`, `sqrt()`, `pow()`.
  These functions are overloaded, so there are versions for `float`, `double` and `long double`.
- `<complex>`
  Common arithmetical operations and functions for complex numbers are provided by this library (or template).
- `<valarray>`
  Provides a template for `valarray` based on arrays.
  These have been optimized for numerical usage, so are more in line with what vectors should be.
- `<numeric>`
  Some numerical algorithms that can accumulate the results of, or generate a sequence from, the operations on one or two sequences. E.g. calculating the inner product of two vectors.

<string>

C++ provides a `string` type as well as some useful operations, such as the string concatenation operator +:

```
string word1 = "Hello";
string word2 = "world";
string line1 = word1 + " , " + word2 + "!\n";
cout << line1;
```

Additional functions provide the ability to compare or to swap strings.

Containers

Containers are objects that store other objects, and we mention them here because the C++ `vector` class is an example of one, provided by `<vector>`.

However, this does not provide what a mathematician would think of as a vector (that is best served by `<valarray>`).

The `vector` class provides a container whose objects are accessible by an index.
Like other containers such as `list`, `queue` and `stack`, this allows for the insertion and deletion of objects from it (consider what would be required to do this with an array).

Iterations on containers are provided by `<iterator>`.
Algorithms for manipulating and operating on containers are provided by `<algorithm>`.

<ctime>

This header defines various functions connected with time, such as `time()`.
In particular, it can provide a very useful benchmarking functionality for your programs.

```
double duration = 0.0;
clock_t start_time = clock();
// Code to time
clock_t stop_time = clock();
duration = static_cast<double>(stop_time - start_time)
          / CLOCKS_PER_SEC;
```

Note, we use `clock()` here, and convert the result to seconds with the `CLOCKS_PER_SEC` constant, rather than the calendar date function `time()`.

<cstddef>

This defines the `ptrdiff_t` and `size_t` types – the latter being the type returned by the `sizeof()` operator.
Both of these types are actually aliases for existing fundamental types on the system (usually `int`) – defined by the `typedef` specifier.

```
typedef double PRICE;
PRICE s0, s1, final;
```

Note that the `sizeof()` operator can be very useful in determining the memory footprint of your program.
As well as giving you the size of the fundamental data types, you can also use it to find the size of arrays too:

```
double A[2][3][4];
cout << sizeof(A) << '\n';  // Returns (2*3*4)*8 = 192
cout << sizeof(A[2]) << '\n'; // Returns (3*4)*8 = 96
cout << sizeof(A[2][3]) << '\n';  // Returns 4*8 = 32
```

Some tips

- Do not begin unless you have a well-defined method/algorithm - write out program in pseudo-code.
- Document everything - program function, method, sources, references, debugging statements
- Do not try to optimise your program *until* you have a program!
- Verify, whenever possible, your programs' results.