Introduction to C++

Practical 2 (Lectures 3-4)

1. Getting used to pointers and references:
   (a) Declare a `double` called `d` with the value `5.0`
   (b) Create a pointer to a double (`double*`) called `p_d` and assign it the address of `d`
   (c) Print out `d` and `p_d`. What does the value of `p_d` mean?
   (d) Create a new double and assign it the value `1.0 + d`
   (e) Create a new double and assign it the value `1.0 + *p_d`
   (f) Print out those two new variables

2. Writing a function:
   (a) Write a function that does anything you like. Declare the function prototype above `main`, and define the function below `main`.
   (b) Delete the prototype, and move the definition above `main`. It should still work as expected.

3. Write code that sends the address of an integer to a function that prints out the value of the integer. Change the value of the integer and verify that the original integer is updated outside your function.

4. Write a function that accepts two floating point numbers, and swaps the values of these numbers.
   (a) Write this function using pointers
   (b) Write this function using references

5. Write a function that returns the scalar (dot) product of two `std::array<double,3>` vectors. Overload this function to multiply two scalar `double` values.

6. The p-norm of a vector $\mathbf{v}$ of length n is given by

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{n} |v_i|^p\right)^{1/p}$$

   where p is a positive integer. Write a function to calculate the p-norm of a given `std::array<double,3>`, where p takes the default value 2. Now template your function to enable it to take and compile-time length N, i.e. `std::array<double,N>`.

   Hint: the definition of `std::array` is

```
template<class T, std::size_t N> struct array;
```

7. Now write the same p-norm function as a C++ lambda function for the specific case of a `std::array<double,3>`. Try inputting p to the lambda function as
   (a) an argument, or
   (b) a capture variable.

8. Overload the p-norm function in Q4 to take a `std::vector<double>`. Loop over the vector using
   (a) an index-based loop,
   (b) a range-based loop,
   (c) an iterator-based loop,
   (d) the `std::accumulate` STL algorithm (in the `<numeric>` header)

9. Write a function multiply that may be used to multiply two matrices, given their sizes. You are free to choose any type to represent your matrices, but you might want to try either a `std::vector<double>` or a `std::vector<std::vector<double>>`

10. Implement the same matrix multiply in Eigen (http://eigen.tuxfamily.org) and time how long your function takes compared with Eigen for a relatively large matrix (hint: look up `std::chrono::high_resolution_clock`, http://www.cplusplus.com/reference/chrono/high__resolution__clock/now). Try to improve the speed of your function as much as you can (hint: google "matrix multiply optimisation", and try tiling or blocking techniques)