

# Introduction to C++

## Practical 2 (Lectures 3-4)

- Getting used to pointers and references:
  - Declare a `double` called `d` with the value 5.0
  - Create a pointer to a double (`double*`) called `p_d` and assign it the address of `d`
  - Print out `d` and `p_d`. What does the value of `p_d` mean?
  - Create a new double and assign it the value `1.0 + d`
  - Create a new double and assign it the value `1.0 + *p_d`
  - Print out those two new variables
- Writing a function:
  - Write a function that does anything you like. Declare the function prototype above `main`, and define the function below `main`.
  - Delete the prototype, and move the definition above `main`. It should still work as expected.
- 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.
- Write a function that accepts two floating point numbers, and swaps the values of these numbers.
  - Write this function using pointers
  - Write this function using references
- 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.
- 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;
```

- 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
  - an argument, or
  - a capture variable.
- Overload the  $p$ -norm function in Q4 to take a `std::vector<double>`. Loop over the vector using
  - an index-based loop,
  - a range-based loop,
  - an iterator-based loop,
  - 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](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)