Chapter 1 Exercises

Learning outcomes

At the end of this lab you will be able to:

- Write a function declaration and function definition.
- Pass arguments including primitive types and arrays to functions.
- Explain the relationship between pointers and arrays
- Create and compare C++ strings (type `std::string`)

Q1. Write a program to input a floating point number and output a rounded version of that number. Provide a function `round(float num)` that returns the closest integer to the argument num. Round upwards to the nearest integer for .5, otherwise round down.

Here is an example of how you might write a function in C++.
The program below is a simple example of the usage of functions in C++. The program uses a function to calculate the factorial of a number entered by the user.

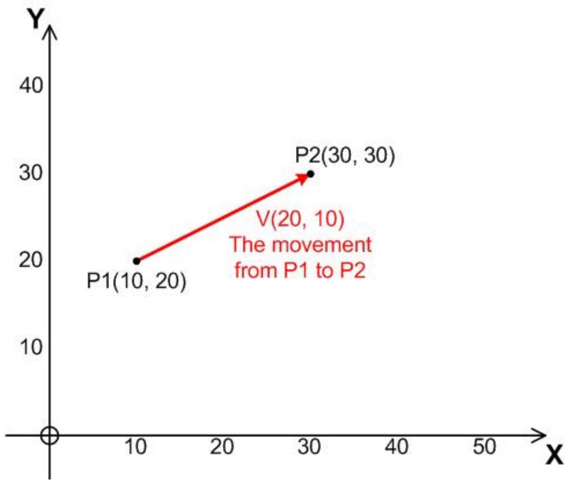
```
/// <summary>
/// @mainpage Factorial - demonstrates basic functions in C++
/// @Author Ross Palmer
/// @Version 1.0
/// @brief Program to calculate the factorial of a number.
///
/// (More detailed comments here if required).
///
/// Date & time of each session and time taken.
/// Total time taken on this project e.g.
///
/// 2/5/16 14:22    40min
/// 4/5/16 16:30    30min
/// 5/5/16 9:00    140min (2hr 20min)
/// 6/5/16 14:00    10min
///
/// Total Time Taken 3hr 40 min
/// </summary>

#include <iostream>

double factorial(int num);

int main()
{
	int num;
	std::cout << "Input a number: ";
	std::cin >> num;
	if (num < 0)
	{
		std::cout << "Cannot compute the factorial of a negative number"
			  << std::endl;
	}
```

```
        else
        {
                std::cout << "Factorial: " << factorial(num) << std::endl;
        }

        system("PAUSE");
}

/// <summary>
/// Computes the factorial of the specified number.
/// </summary>
/// <param name="num">A positive integer</param>
/// <returns>The factorial of num</returns>
double factorial(int num)
{
        if (num <= 1)
        {
                return 1;
        }

        double sum = num;
        for (; num > 1; num--)
        {
                sum = sum * (num - 1);
        }

        return sum;
}
```

Q2. Write a program to input a value in degrees and output the result in radians. Provide a function `deg2rad(double degrees)` that returns a double value in radians. Note that `radians = degrees * π / 180`
C++ does not provide a constant for π in the math library. You can calculate a value for π instead. Hint: the `acos(x)` function (arc cosine) returns a value in the range 0 to π where x is in the range -1.0 to 1.0

Q3. Shortly, you will write a function to normalise a vector (i.e. create a unit vector). First, a quick recap on vectors.

Vectors are can be shown in 2D where we only need (X, Y) values. Look at the diagram below:



You can see the positions (or points) P1(10, 20) and P2(30, 30). You can also see that the arrow of *movement* from P1 to P2 can be represented as the vector V(20,10). Notice how the values in the vector are found by subtracting the source point coordinates from the destination point coordinates. This means that the direction of the vector is important: the vector in the reverse direction, from P2 to P1, is (-20, -10). So, in general:

The vector V from point P1(X_1, Y_1) to P2(X_2, Y_2) is V(X_2 - X_1, Y_2 – Y_1).

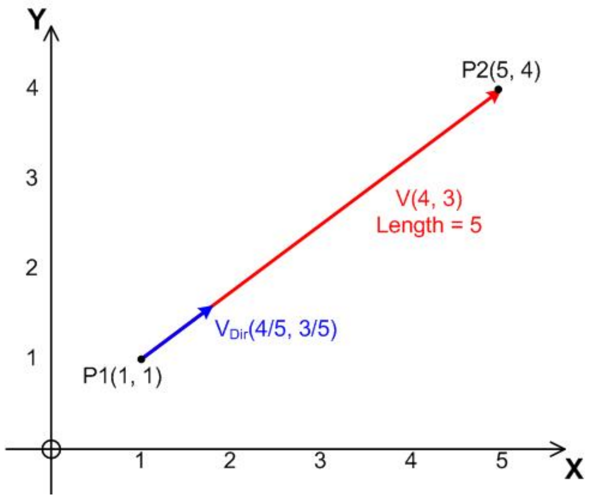
Direction of Vectors

Vectors can also be used to represent a direction (in the sense of North or North-West). To do this we create a vector that 'points' in the direction we want, and has a length of 1 unit.

We often need to find the direction vector that 'points' from one position to another. The process we use is:

1. Get the vector between the two positions V(X, Y, Z)
2. Find the length of this vector L
3. The direction vector is V_{Dir}(X/L, Y/L, Z/L)

This process is referred to as 'normalising' a vector. Here is a 2D example of the process:



Direction vectors are useful when we want to move something by a given distance in a particular direction. We multiply the direction vector by the distance we want to move. For example, in the diagram above, if we want to move by 12 units in the direction V_{Dir}:

Movement vector = 12 x V_{Dir} = (48/5, 36/5) = (9.6, 7.2)

Write a program with a function

```
void vectorNormalise(float inputVector[], float resultVector[])
```

that normalises `inputVector` and puts the result in `resultVector`. Clearly `inputVector` and `resultVector` are both 2-element arrays. They might be declared in function `main()` like this:

```
// f denotes a constant of type float
float points[] = {1.1f, 2.2f, 3.3f, 4.4f};
float result[2];    // An uninitialised two element array
```

Q4. Write a program to input 10 names into an array of strings. Create another array of pointers such that the first pointer holds the address of the first string, the second pointer the address of the second string and so on.

Call a sort function with the following prototype:

```
void sort(string *pNames[]);
```

Sort the array of pointers as described in lecture 4. Output the list of sorted names by traversing the array of pointers.

Notes:

C++ supports a string class. String objects can be compared using the `compare` method (http://en.cppreference.com/w/cpp/string/basic_string/compare), they can also be compared using the standard relational (<, >, >=, <=) and equality operators (==, !=). Here's a quick example:

```
#include <iostream>
#include <string>

int main()
{
	std::string names[2];

	std::cout << "First Name ? ";    // input AAA
	std::cin >> names[0];

	std::cout << "Second Name ? ";   // input AAB
	std::cin >> names[1];

	std::cout << names[0] > names[1]; // outputs 0 (false)
}
```

To see the full set of functions supported by strings, go to:
http://en.cppreference.com/w/cpp/string/basic_string