Object Oriented Programming
Laboratory 2 - Functions & Arrays

1 Aims & Objectives

The aims of this lab are:

- to emphasise the value of functions in programming;
- to understand function prototypes;
- to learn to pass variables into functions;
- to learn how to pass variables out of a function;
- to understand the nature of how C and C++ import variables into functions;
- to examine the declaration and use of arrays, and to explore the power of arrays combined with loops;
- to learn how to initialise arrays, and how to declare multidimensional arrays.

2 Functions

We begin by looking at functions.

2.1 Calling functions

Type in the following program.

```
1 #include <iostream>

  using namespace std;

  // Function prototypes
6 int main(void);
```

```
void HelloWorld(void);

// Now the function itself. Note the lack of a semi-colon
void HelloWorld(void)
{
  cout << "Hello␣World\n";
}

int main(void)
{
  int loop;

  // When you call a function you must
  // use the brackets after it (), even if there
  // are no parameters
  for(loop=0; loop<5; loop++) HelloWorld();
  return(0);
}
```

2.2 Passing arguments to functions

Consider the following function

```
void AddOne(int number)
{
  // We don't declare variables that are incoming
  // here because they are already declared in the
  // function prototype...

  number++;

  cout << "Number␣is␣" << number << "␣(2)␣(AddOne)\n";
}
```

This function when called with `AddOne(3)` for example assigns the variable `number` to be 3, adds one to the variable to obtain 4, and prints it out to the screen.

Add this function to the following listing

```
int main(void);
void AddOne(void);
```

```
int main(void)
{
  // This variable is local to main only...
  int number = 2;

  cout << "Number␣is␣" << number << "␣(1)\n";
  AddOne(number);
  cout << "Number␣is␣" << number << "␣(3)\n";

  return(0);
}
```

This program isn't totally complete, can you complete it? Run the program, you will observe that the change to number is not permanent. This is because a copy of number is passed into the subfunction and not the original value.

2.3 Returning values

Alter the `AddOne()` function as shown below.

```
int AddOne(int number)
{
  // We don't declare variables that are incoming
  // here because they are already declared in the
  // function prototype...

  number++;

  cout << "Number␣is␣" << number << "␣(2)␣(AddOne)\n";

  // The variable number is about to be destroyed,
  // provide a mechanism for passing the changed value
  // out...
  return(number);
}
```

Make any other necessary changes to the program that are required to compile it. Does this result in the value of number changing on the third line? If not is it possible to effect the change?

2.4 Recursive functions

We looked at a function to produce a factorial of a number. Can you write a function with prototype

```
unsigned int Factorial(unsigned int n);
```

that produces the factorial of a number and that *is not* recursive.

3 Arrays

We now turn our attention to deal with arrays.

3.1 Basics

To declare an array of 10 integer variables we can use the following

```
int variables[10];
```

Write a program that prompts the user for 10 integer variables and gives the average of them all.

3.2 Multi-dimensional arrays

The declaration `int matrix1[3][3]` can be used to declare a two dimensional array of numbers. Write a program that creates such an array and prompts the user for all the values, and then prints them back out.

Now declare a second array `matrix2` and fetch the values from the user in the same way.

Finally create a third array `matrix3` and fill it with the result of multiplying `matrix1` by `matrix2`.