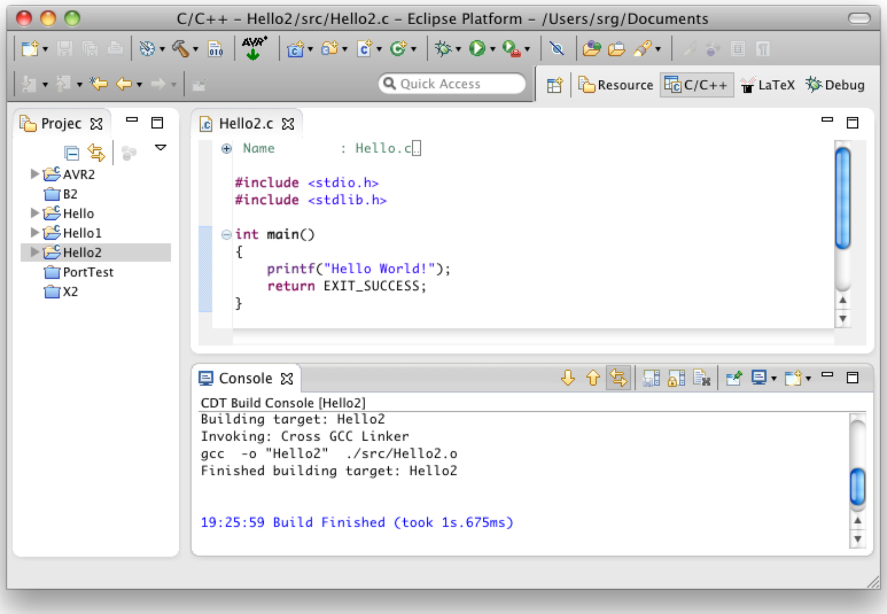
C3

Functions & Pointers

One of the challenges of writing larger programs in C is to make the code easier to read and understand by another programmer. To help with this, C has a number of features to aid in breaking up the code into smaller modules with clearly defined interfaces. In this exercise you will learn to use the principal feature which is the function. Additionally, you will learn about pointers which enable you to pass variables by reference to functions, enabling the function to update the values and returned these updates to the point of the code that called the function. Finally, multiple header and source files are introduced as a further way to add modularity to your code.



```
 Name         : Hello.c

#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello World!");
    return EXIT_SUCCESS;
}
```

```
CDT Build Console [Hello2]
Building target: Hello2
Invoking: Cross GCC Linker
gcc  -o "Hello2"  ./src/Hello2.o
Finished building target: Hello2


19:25:59 Build Finished (took 1s.675ms)
```

Schedule

Preparation Time	:	3 hours
Lab Time	:	3 hours

Items provided

Tools	:	
Components	:	
Equipment	:	
Software	:	`gcc`, `Eclipse`

Items to bring

Course Textbook - *C Programming in Easy Steps*

Identity card

Laboratory logbook

Version: September 29, 2013

Enrico Costanza, Steve R. Gunn
Electronics and Computer Science
University of Southampton

Before entering the laboratory you should read through this document and complete the preparatory tasks detailed in section 2.

> **Academic Integrity** – *If you wish you may undertake the preparation jointly with other students. If you do so it is important that you acknowledge this fact in your logbook. Similarly, you will probably want to use sources from the internet to help answer some of the questions. Again, record any sources in your logbook.*

You will undertake the exercise working with your laboratory partner. During the exercise you should use your logbook to record your observations, which you can refer to in the future – perhaps to write a formal report on the exercise, or to remind you about the procedures. As such it should be legible, and observations should be clearly referenced to the appropriate part of the exercise. As a guide the ✏ symbol has been used to indicate a mandatory entry in your logbook. However, you should always record additional observations whenever something unexpected occurs, or when you discover something of interest.

For each Task you should create a new directory so that you have a working version of every program at the end of the lab. Remember to place comments in your code.

You will be marked using the standard laboratory marking scheme; at the beginning of the exercise one of the laboratory demonstrators will mark your preparatory work and at the end of the exercise you will be marked on your progress, understanding and logbook.

Notation

This document uses the following conventions:

✏	An entry should be made in your logbook
☆	A hint to application areas—can be ignored

1 Introduction

This lab introduces you to programming with functions and pointers. These are two of the most fundamental concepts in C programming.

1.1 Outcomes

At the end of the exercise you should be able to:

- Design and implement simple C programs where you declare, define and call functions.
- Design and implement simple C programs where you use pointers to manipulate variables through functions.

2 Preparation

Read chapters 6 and 7 of the course textbook (McGrath, M, "C Programming in Easy Steps," In Easy Steps Ltd., 2012). Then complete the following exercise.

1. Analyse program arr_func.c. Write notes in your logbook about how the array was passed to another function by `main` and also on how the array was actually used in that function. Comment in particular on the use of the keyword `const` in this program.
2. Write a function that receives a pointer to variable of type `double`, sets the variable value to zero and returns.
3. Write a function that receives a pointer to an array of type `double`, together with its size, and sets all element values to zero, then returns.

You are not required to compile and test the preparation programs, but if you can, why don't you try that too?

3 Laboratory Work

Open the Eclipse application on the computer you are using and when asked for the workspace you wish to use, make sure the checkbox for *use as default* is not ticked and type `H:\ELEC1201\Labs\C3`.

Remember to create a new C project (File -> New -> New project.. -> C/C++ -> C Project), for each task of the exercise and name it with the appropriate part of the exercise for easy future reference, e.g. `C3_3.1.1`. You can select "Hello World ANSI C Project" as "Project type" and replace the contents with your own code, or create an "Empty Project" and add in a new source file.

Remember, if you are asked if you want to "Switch perspective" answer positively.

3.1 Functions and Arrays

3.1.1 Passing an array to a function

1. Write a program where you declare an array of 10 floating point values, you print the array and you calculate (and print) its average value.

2. Modify the program above so that the printing of the array and the average calculation take place in two separate functions. These functions can be named `print_array` and `average_array`, they should receive as argument a pointer to the array and the size of the array (as an int).

Hint: you can start from one of the programs that you debugged in the previous lab.

3.1.2 Filling arrays with random numbers

Compile and run program userand.c. Does it produce a set of numbers that look random? Run it again. Is the series of random numbers the same? Why is this? Add `#include <time.h>` at the top of your program, add the call `srand( (unsigned)time( NULL ) );` just before the for loop. Run it again. Is there a difference? Look on the Internet for information about the `rand()` and `srand()` functions.

Write a function that fills an array of integers with random values. Write a program to test and demonstrate your function in action. Hint: you can start by modifying program arr_func.c from this lab preparation.

Pseudorandom sequences are used in chip testing, spread spectrum communication, and to determine the transfer function of electronic and acoustic systems.

3.2 Working with multiple source and header files

In chapter 7 of the book you read about distributing functions across multiple `.h` and `.c` files, and how to compile these files into one executable using command line GCC. How can we do the same in Eclipse?

From the "File" menu select New -> Source File or New -> Header File. A dialog window will pop-up: Click the button browse and select the "src" folder within the project you are currently working on (a note of warning: eclipse will show here all the projects that are currently open, make sure you select the correct one); fill in the name of the new header (e.g. `array_utils.h`) or source (e.g. `array_utils.c`) file; select "Default C source template" (not C++!); click on the "Finish" button.

Create a new project and modify one of the programs above so that the extra functions are placed in separate source and header files (just one extra source and one extra header). Compile and run your program using Eclipse. Make a note of the procedure in your logbook.

3.3 Implementing an Existing Algorithm: Bubble Sort

Implementing algorithms described in general terms or implemented in a language different from the one you need to use is a very common task in real life. At this stage you should have enough knowledge of the C language to try implementing a simple algorithm to sort things, based on the description you can find on Wikipedia: the bubble sort algorithm. Please note that the standard C libraries include a function called `qsort`, which implements the quick sort algorithm. We will look at the `qsort` function in a later lab. Note that the quick sort algorithm is more efficient than bubble sort, but bubble sort is a lot easier to implement, and that is why it was chosen.

Write a program to sort the elements of an array into ascending order using the Bubble sort algorithm. The sorting should take place in a separate function:

```
void sort(const int size, const float *input, float *output)
```

Your function should not modify the content of the input array (that's why the keyword `const` is there), and it should replace the content of the output array with the sorted content of input. Create also a function to display the arrays and use it to display both the input and the output. (or copy it from one of the examples above.)

Hints: Even though there is a lot of information on the Wikipedia page about bubble sort (link above), you need to focus only on the general description (first few lines of the article, up to the table of contents) and on the "implementation" section. Moreover, do not worry about optimising the code. You can use an int (or even better char) variable and store in it 1 or 0 to mean logical true or false. When you are done, try to pass two arrays of different size, in particular make the output array both smaller and bigger than the input array, and note in your logbook what happens.

4 Optional Additional Work

4.1 Matrices as Bidimensional Arrays

Write a program that converts 1-dimensional arrays to 2-dimensional arrays and manipulates them. Write functions for all operations that you are required to do (e.g. display a mono-dimensional array, display a multi-dimensional array, horizontalFlip, etc..).

One way to pass a multi-dimensional array to a function without getting compiler warnings, is to use the following two arguments:
`const int msize, int (*ma)[msize]`
For example:
`void showMatrix(const int msize, int (*ma)[msize]);`
This is because one-dimensional arrays get passed to functions as pointers, however, two-dimensional arrays are not equivalent to double pointers. For more information see for example: `http://stackoverflow.com/questions/4434395/c-passing-dynamically-sized-2d-array-to-function`
Do you find this notation confusing or ugly? Well, part of the point of this exercise (as well as the next

one) is to show you that often you can avoid using two-dimensional arrays and use one-dimensional arrays instead!

Hint: When you need a function to return an array, pass an empty array as an argument and fill it with the result.

1. Create an array of 25 elements `a` (whatever type you like), fill it with random elements and display it on screen.
2. Convert the array into a 5x5 matrix `ma` (2-dimensional array) and display it.
3. Create a second 5x5 matrix `mb` which is the horizontally flipped version of `ma` (i.e. the first column of `ma` is the last column of `mb`) and display it on screen.
4. Create a third 5x5 matrix `mc` which is the vertically flipped version of `mb` (i.e. the first row of `mc` is the last row of `mc`) and display it on screen.
5. Convert `mb` back to a mono-dimensional array `b` and display it on screen.

4.2 Matrices as Monodimensional Arrays

1. Repeat exercise 3 from section 4.1 (horizontally flip the 5-by-5 matrix) starting from `a` without using multi-dimensional arrays and display the result on screen.
2. Accessing array elements through the `[]` operator is slower than accessing them by moving pointers around. Repeat the previous exercise using pointers, rather than array indexes.