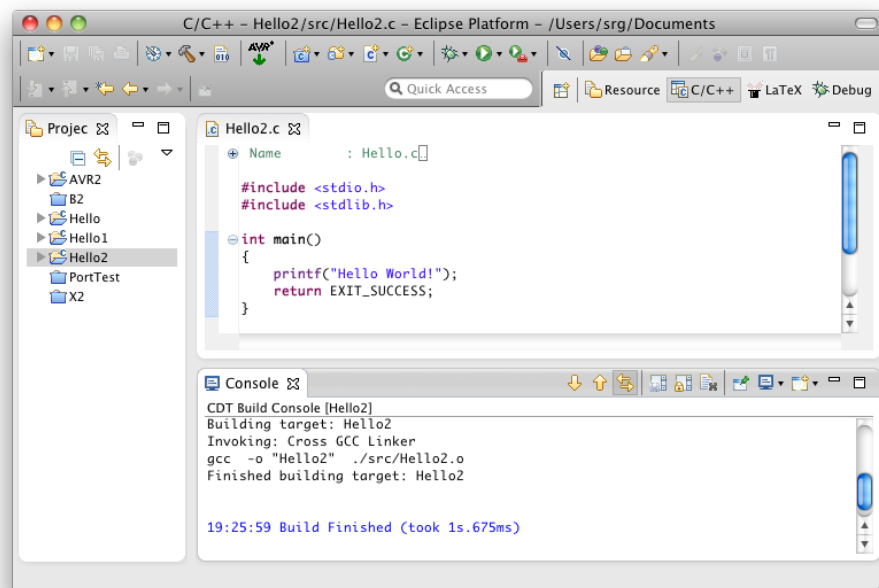


C2

Operations, Conditions & Loops



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: October 4, 2013

©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 loops and conditionals. It also introduces you to two debugging methods for finding errors at runtime in your programs.

1.1 Outcomes

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

- Design and implement simple C programs using `if` statements, `while` and `for` loops to solve simple mathematical problems.
- Debug programs using `printf` and the GDB debugger integrated in Eclipse.

2 Preparation

Read pages 42-47 of chapter 3, then all of chapters 4 and 5 of the course textbook (McGrath, M, "C Programming in Easy Steps," In Easy Steps Ltd., 2012). Then complete the following exercise.

Examine the following five C programs:

Single For Loops `countdown.c` `centfahr.c` `objdrop.c`

Nested For Loops `loopnest.c` `addtable.c`

Can you understand what is going on? In particular note, in your log book:

1. the starting conditions
2. the increment and how it is achieved
3. the test involved in controlling each loop.



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\C2`.

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. `C2_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 Loops

Write a program which displays n^2 for the first N integer numbers.

3.2 Loops and Conditional Expressions

Write, compile and run a program to generate and display n^2 for even numbers and n^3 for odd numbers, for the first N integer numbers..

3.3 A simple debugging technique

The program `average_bug.c` contains a bug. Create a new project for it, copy the code into it, compile it and run it. As all the values in the array are between 1 and 5, the average should be within that range too. However, the program should give you a different result.

One way to find and fix bugs is to look at the intermediate results calculated by the program. You can do this using `printf`. For example, try to print out the values of each element of the array inside the loop, by adding a `printf("%d\n", data[i]);` Are those values what you expect? Look at the values of the `avg` variable, as you add to it. Are those what you expect?

You may find the bug just by looking at the code, given that this is a simple example. The point here is about the technique of looking for bugs:

- ▶ consider each portion of your program individually;
- ▶ think about what you expect to happen there;
- ▶ print out the values of your variables to see if things go as you expect.

Take notes in your log book about the debugging process and about the bug that is in this example.



3.4 A more advanced debugging technique

A more advanced and general way to debug code is to use a debugger tool. A very popular and powerful tool is called GDB, like GCC it is open source. GDB has a command line interface, but that is not very convenient to use, instead we will use it via Eclipse, given that it is integrated in the IDE.

You will now debug `min_max_bug.c` using this debugging technique. Please create a new project for it, copy the code into it, compile it and run it. You should notice that the minimum value calculated is zero, while it should be one. There is a simple bug in the program, once again this is just an example, so even if you can guess what the bug is, please go through the debugging technique, which will be useful in the future, to find real bugs in more complex programs.

When debugging code the challenge is to locate the area of the problem and look carefully at how the code is behaving in that region. Setting breakpoints is a simple way to get your program to execute and then stop in the region of interest. Then you are able to explore. On the left of each line in the

eclipse editor, you are able to set breakpoints. Right-click to the left of the line in the source file in Eclipse where you wish to insert a breakpoint, and select the first option "Toggle Breakpoint".

Set a breakpoint on the line `if (max < data[i]){`. Then go to the "Run" menu and select "Debug". A pop-up window may ask you if you want to change perspective, answer yes. If you do not get the pop-up window manually change perspective from Window -> Open perspective -> Other -> Debug.

Once the debug perspective is active, go again to the "Run" menu and click on "Resume" – you should see that line 12 of your code is highlighted and there is an arrow on the left of the line.

In the debug perspective, on the top part of the screen, you have a lot of information to help you understand what state your program is in. On the top left you can see the "Stack trace" indicating the sequence of nested functions called so far in the program. In this case it is just main, inside the only thread of your program. This will become more useful when we will work with more functions, besides main.

On the top-right you should see the "Variables" view, a panel showing you all the variables declared in the current local scope of your program and their respective values. In the same panel you should see a few other tabs. The "Breakpoints" tab lists the breakpoints you set in the program and allows you to remove them or disable them. For the time being do not worry about the "Registers" and the "Modules" tabs. Another useful view is the "Expressions" view, if it is not already visible, you can add it by going to the "Window" menu and selecting Show View -> Expressions. The Expressions view is somehow similar to the the variables view, but it allows you to monitor any variable-based expression you manually enter in it.

The debugger allows you to run your program step-by-step, stopping at each break-point that you choose. Once the execution of the program stops at a break point you can:

Step Over to the next instruction by selecting Run -> Step Over

Step Into if the current instruction is a function call, you can step inside the function by selecting Run -> Step into

Step Out you can also step out of a function selecting Run -> Step out (you will not need "Step Into" or "Step Out" until we learn more about functions).

Resume resume the execution and go to the next break point by selecting Run-> Resume

Using the Variables view, the Step Over and the Resume functions, and if necessary more breakpoints, analyse the execution of your program. Take notes in your log book about the debugging process and about the bug that is in this example.




3.5 Calculations for an RLC Circuit

An oscillatory circuit can be constructed from a number of passive components, a resistor (R), an inductor (L) and a capacitor (C). In one particular configuration the ratio of the voltage across two

parts of the circuit may be written as

$$\frac{1}{\sqrt{(1 - \omega^2 LC)^2 + \omega^2 R^2 C^2}}$$

where ω is the angular frequency ($\omega = 2\pi f$) and is given in radians/sec. Write down this expression in your logbook in the form required in C programs. 

The values of the components in our particular circuit are: $L = 30\text{mH}$, $R = 2.2\text{k}\Omega$, $C = 10\text{nF}$.

Write a program in C, using a for loop, which calculates the above voltage ratio for $20\text{k} \leq \omega \leq 250\text{k}$ in steps of 20k radians/sec and prints out the results clearly e.g.

Freq (Rad/s)	Ratio
0.00	1.00
0.20	1.04
⋮	⋮

After printing all the values your program should also print out the value of ω for which the voltage ratio is maximum (this will require you to use an if statement and one or more extra variables). To print out each line you can use the following statement: `printf("%.2f\t\t%.2f", omega, ratio);` where `omega` and `ratio` are variables.

Remember to include the `stdio.h` file at the top of your program so that the compiler understands the syntax for the `printf` function.

Many mathematical functions are available in C, including the square root. To use them, simply include the `math.h` file at the top of your file.

Start by writing a program that calculates the voltage ratio for a fixed ω value, then modify this to use a for loop, finally modify it again to calculate the ω corresponding to the maximum.

4 Optional Additional Work

4.1 Adding Simple Graphics

Try adding some elementary graphics to the program you wrote in Section 3.6. Create a new version of your program that outputs something like this:

```
0.00  1.00  =====
0.20  1.04  =====
0.40  1.16  =====
0.60  1.41  =====
0.80  1.86  =====
1.00  2.00  =====
1.20  1.34  =====
1.40  0.84  =====
```

```
1.60 0.57 =====
1.80 0.41 =====
2.00 0.32 =====
2.20 0.25 =====
2.40 0.20 =====
2.60 0.17 =====
2.80 0.14 =====
3.00 0.12 =====
```

Hints: Multiply the ratio by 25 and output that many = on the same line (you will need a loop inside your original loop to print the =). You can use a simple cast to truncate the floating point values and convert them to integers, if you want more accurate results, you can use the `round()` function, to use it you need to also add `#include <math.h>` at the beginning of your program. For more information about the math functions, see the reference section at the end of the textbook.

4.2 An Application of Loops: the Newton-Raphson Method

The formula for finding the roots of an equation ($f(x) = 0$) using the Newton-Raphson method is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

where f' indicates the derivative of the function.

Write a program, using loops, to find the roots of $f(x) = 2x^2 - 3$.

Think carefully about the stopping condition you should use.

More information about the [Newton-Raphson method is available on Wikipedia](#)