Lab 04 – C Language Memory Model

This lab explores the C language memory model and how pointers may be used to manage memory. Throughout this lab, you will review some programs in C as well as compile, test, debug and fix them.

By the end of this lab, you will be able to:

- Use the C language to express code solutions to simple problem specifications;
- Understand the memory model of the C language;
- Use C constructs respecting the memory model of the C language;
- Test, debug and fix source code in C;
- Generate executable code from source code in C using the gcc tool.

Copy the lab files into a new directory

Make a Lab4 directory in your account, enter it, and copy over the files from the source, which are located in the lab environment in the **`/home/rbittencourt/seng265/lab-04/`** directory.

Pointer tracing exercise: one_function.c

The program **`one_function`** uses integer variables, and pointers to those variables. Try tracing the code, without compiling it (you should use pencil and paper for this sort of code tracing).

Can you predict the output? (You won't be able to predict the exact output of the line **`printf("%p\n", (void *)pp);`**, since it prints a memory address that might vary from one run of the program to the next).

Compile the program, and run it - was your code-traced prediction correct (again, except for the last line)?

Array bounds exercise: danger.c

This program creates an array of size 5, named **`nums`**. It also declares a pair of variables, which may be placed just before or just after the array in memory.

Everything works fine as long as we are careful to respect the array boundaries. However, the function **`rogue()`** modifies the array, and does so in a way that overwrites the other two variables.

Compile and run the program, and observe the results. Can you explain the values you see on the line below the dashes?

Programming exercise: q_array_rotate.c

This program, once completed will include a function named **`rotate()`**, which shifts the values in an array to the left, by the number of spaces provided as an argument. For example, if an array starts out with these values:

`1,2,3,4,5,6,7,8,9,10`

And **`rotate()`** is called with the number 1 as its argument, the array after the function call should contain the values:

`2,3,4,5,6,7,8,9,10,1`

Note that the first element of the array has 'wrapped around' to the end of the array. You'll need to write the **`rotate()`** function, and you'll also need to write the function calls themselves in **`main()`**. As you do so, think about the kinds of information **`rotate()`** will need. How will you tell it which array to rotate? Do you need to tell it the length of the array? Your solution should work for any value of N >= 0, even values much larger than the length of the array.

For an extra challenge, make **`rotate()`** work with: - arrays of any length - negative values for the number of spaces to rotate (this should result in rightward rotation).

Programming exercise: q_word_freq.c

This program creates a pair of strings full of ‘words’ - whitespace separated sequences of characters. It also accepts an argument at the command line – the ‘word to search for’.

The goal of the exercise is to complete the function named **`word_freq()`**, which takes a string and this single word, and returns the number of times the word appears in the string.

For example, for the first string st0, the word **`match`** appears four times, so if **`word_freq(st0, "match")`** is called, it should return the number 4. Case is significant, so for the purposes of this program **`cookie`**, **`Cookie`** and **`COOKIE`** are all different words, and so a search for one of them should only match to that single, identical word.

Hint: The following c functions may be helpful: **`strtok()`**, **`strlen()`**, **`strncmp()`**.

Present your results to your TA

Take a few minutes to present the results of your code changes and creation to your TA. That is how you will be assessed in this lab.