Course Name: Comp Arch Lab

Course Number and Section: **14:332:333:01**

Experiment: Experiment # 2 – ***Introduction to C Programming Language***

Lab Instructor: Ali E. Haddad

Date Performed: 9/24/18

Date Submitted: 10/08/2018

Submitted by: [Abdullah Ghani 171009840]

Electrical and Computer Engineering Department
School of Engineering
Rutgers University, Piscataway, NJ 08854
ECE Lab Report Structure

1. Purpose / Introduction / Overview – describe the problem and provide background information

The purpose of this lab was to introduce us to the C programming language. We were given a few .C files and were assigned to debug and compile them.

2. Approach / Method – the approach took, how problems were solved

We read through the lab manual which itself was pretty explanatory. Having a background in java programming really helped make the process make more sense and easier to grasp. The instructor was always ready to guide us through any problems that we faced.

3. Results – present your data and analysis, experimental results, etc.

Exercise 1:

1. Explain the changes you made.
   a. I have included the simple.c file in the repo commenting the changes I have made

2. Explain the minimum number of distinct values needed for the preprocessor macros.
   a. The minimum number of distinct values needed are 1 because it provides the required result fitting in with the requirement of the global variables

3. What does the -o flag do with gcc?
   a. Using gcc with the `-o`, specifies the name of the executable file that `gcc` creates. Using `-o`, you would use the following commands to compile `program.c` into a program named `program`, and then run it. This is helpful if you don't want all of your executable files to be named `a.out`.

Exercise 2:

1. Explain how do you set the breakpoint at main, and how you run up to that breakpoint.
   In **gdb**, you can set a breakpoint on a method or function name:
   (gdb) **break main**
   step executes one instruction. It will step into functions. Keep stepping until the breakpoint.

2. A list containing the additional gdb commands.
   Included in the repo as gdb.txt

Exercise 3:

1. Explain the bug and your fix to the function.
   `while (a != NULL) was changed to while( a &&b)`
   `this means the loop breaks if one of the lists is smaller than the other and you don't end up comparing a null element.`
   `Fixed code ll_equal.c is included in the repo.`

Exercise 4:

1. Describe how you run CGDB to completion on the executable created by compiling interactive_hello.c without getting stuck.
   Create a file called name.txt. This file just contains your name.
   When running the C program, run it like this
   ```
   gcc -g -o int_hello interactive_hello.c
   $ gdb int_hello<name.txt
   ```
   This feeds it the input without it getting stuck during debugging

Exercise 5:

1. Implement `ll_cycle.c` with the completed `ll_has_cycle()` function. Comment your code and provide explanation for your solution

When you advance tortoise by one node, a null pointer check is unnecessary because it is already done when you advance hare by 2.

```
int ll_has_cycle(node *head) {
        /* your code here */
node * tortoise = head, * hare = head;
  while (hare && hare->next && hare->next->next) {
    hare = hare->next->next;
    tortoise = tortoise->next;
    if (tortoise == hare)
      return 1;
  }
  return 0;
}
```

You increment the pointers and check if the data the node pointer points to is equal. If yes then the list is cyclic. Else it is not

2. Conclusion / Summary – what was done and how it was done

We got comfortable in understanding pointers in C. We also learnt how to use the debugger to undesratnd the problems in our code and the ease of using it with the command line.