Course Name: Computer Architecture and Assembly Laboratory

Course Number and Section: 14:332:331:01

Experiment: [Lab 2 – Introduction to C Programming Language]

Lab Instructor: Mengmei Ye

Date Performed: 9/28/18

Date Submitted: 10/11/18

Submitted by: [Aryeh Ness, RUID: 173000855]

Course Name: Computer Architecture and Assembly Laboratory

Course Number and Section: 14:332:331:04

! Important: Please include this page in your report if the submission is a paper submission. For electronic submission (email or Sakai) please omit this page.

----------For Lab Instructor Use ONLY----------

GRADE: _____

COMMENTS:

Exercise 1:

1. I changed V0=3, V1=0, V2=1, V3=3
2. There are 4 minimum distinct values needed here, as each value has a direct effect on the output of the function.
3. Opens a new executable version of the same file.

Exercise 2:

1. The command “b hello.c:main” will stop the function at the beginning of the main function. To run up to this point, we use the command “run.”
2. 1) run arg_1 arg_2, where “arg_1” and “arg_2” are the input arguments.
   2) cond n conditional, where n=the breakpoint number, and (conditional)=the condition you want the breakpoint to reach.
   3) n
   4) s
   5) continue
   6) print variable_name
   7) display /f variable_name
   8) info locals
   9) q

Exercise 3:

1. The while loop causes the error, so completely getting rid of it fixes the function and lets it run properly. It’s useless based on the nature of the nodes-you only need to check the first value in the list to see if they’re equal.

Exercise 4:

1. Within the main function, before the function fgets runs, I inputted the following code: “ FILE *fp; fp=fopen("Name","r");” Then, I got rid of “stdin” in the function “fgets,” and I replaced it with “fp.” The main function now executes without any user input, both in and out of debugger mode. “Name” is a file with text in it that is being read as the input.

Exercise 5: Here is the full code for the implemented function:

```
int ll_has_cycle(node *head) {
        struct node *hare=head;
        struct node *tortoise=head;
        while(hare){
           hare=hare->next->next;
           tortoise=tortoise->next;
           if(hare==tortoise){
              return 1;
           }
           if(!hare){
              return 0;
           }
           if(hare->next==NULL){
              return 0;
           }
     }
}
```

First, I created the pointers hare and tortoise to point to the passed-in value for the function, head. Then, I needed a loop that would keep running until it was determined the list was cyclical or acyclical. I used the loop “while(hare),” since this loop would only end once hare was null-a sign that the loop was indeed

acyclical. Then, I had the loop start by moving hare two position spaces ahead, and tortoise one. Three checks were then performed: 1) if hare and tortoise pointed to the same node, the code could return 1==true, and we could conclude the list is cyclical. 2) If hare was ever NULL, we'd know the list was acyclical-shown by "if(!hare)." 3) This check had to be added in specially for list 5, as the above two checks cover all the other lists except for this one. In the event that hare stops at the last defined value in the list, if it tries to move forward two positions at the beginning of the next while loop, it will return a memory out of bounds error, because it's trying to move to a place that doesn't exist. To protect against this, the third check was added to see if the next value directly after hare is NULL. If it is, we know the list is acyclical and 0 is returned. If it's not, the while loop is able to keep properly functioning to determine the list is cyclical.