ASHWIN ANAND - COMPUTER ARCHITECTURE LAB 2

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

Course Name: Computer Architecture Lab

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

Experiment: Lab # 2 – Introduction to C Programming Language

Lab Instructor: Mingbo Zhang

Date Performed: February 20th, 2021

Date Submitted: March 1st, 2021

Submitted by: Ashwin Anand - 192007894

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

Computer Architecture and Assembly Lab
Spring 2021

Lab 2
Introduction to C Programming Language

Goals

1. Learn how to compile and run a C program;
2. Examine different types of control flow in C;
3. Introduction to C debugger;
4. Gain experience using ***gdb*** to debug C programs;
5. Get comfortable working with pointers.

Introduction

C is syntactically very similar to Java, but there are a few key differences of which to be wary:
• C is function oriented, not object oriented.
• C does not automatically handle memory for you.
  – In the case of *stack memory*, local data is garbage immediately after the function in which it was defined returns.
  – In the case of *heap memory* (things allocated with malloc and its related instructions), data is freed only when the programmer explicitly frees it.
  – In any case, allocated memory always holds garbage until it is initialized.
• C uses pointers explicitly. *p tells us to use the value that p points to, rather than the value of p, and &x gives the address of x rather than the value of x.

There are other differences of which you should be aware, but this should be enough for you to start working with C.

Compiling and Running a C program

We will be using an online C compiler to compile and run C programs: https://www.onlinegdb.com/online_c_compiler.

HelloWorld Example:

```
#include <stdio.h>
```

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

```
int main()

{

      printf("Hello World");

       return 0;
}
```

Output:
Hello World

Exercise 1 [18 pts]

Please read and run the following code and answer the questions.

```
#include <stdio.h>

int main(void) {
        int a;
        char *s;
        int v0 = 0, v1 = 0, v2 = 0, v3 = 0;

        printf("Exercise 1:\n====================\n");

        switch(v0) {
                case 0: printf("Hello September\n"); break;
                case 1: printf("Go Rutgers!\n"); break;
                case 2: printf("Busch Student Center \n"); break;
                case 3: printf("New Brunswick \n"); break;
                case 4: printf("Go ");
                case 5: printf("Rutgers!\n");
                default: printf("Have a great semester!\n"); break; }

        for(a=0; a<v1; a++) {
```

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

```
                printf("RU ");
                }
        printf("\n");

        if (v2 == 6) {
                s = "Go";
        }
        else {
                s = "Hello";
        }

          if(v3 != 3) {
                printf("%s RUTGERS!\n",s);
          } else {
                printf("%s Scarlet Knights!\n",s);
          }

          return 0;
}
```

1. [10 pts] What is the output of the program? Please explain why.

The output of the program is:

Exercise 1:

==========

Hello September

Hello RUTGERS!

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

The first printf statement prints a new line and then the string “Exercise 1:” and then prints a new line.then string “==========” is printed. Then a new line is printed.

The switch statement is a conditional instruction which will print based on the value of the variable v0. Since v0 is set to 0, the string “Hello September” is printed and then a new line is printed.

Next, the for loop will not be executed as the start and end conditions are met with variables v and a equal to 0. A new line is printed after exiting the for loop.

In the next IF conditional statement, “Hello” gets printed as variable v2 not equal to 6.

Next since the value of variable v3 is not equal to 3, the strong “Rutgers” is printed. Then a new line is printed.

Lastly the program exits.

2. [8 pts] If we need to get the following output, what are the values of v0, v1, v2, and v3?

```
Exercise 1:
==========
Go Rutgers!
Have a great semester!
RU RU
Go Scarlet Knights!
```

The values for v0, v1, v2 and v3 should be as follows:
v0 = 4
v1 = 2
v2 = 6
v3 = 3

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

Exercise 2 [22 pts]

This section is intended for students who aren't familiar with what debuggers are. A debugger, as the name suggests, is a program which is designed specifically to help you find bugs AKA logical errors or mistakes in your code (side note: if you want to know why errors are called bugs, look here). Different debuggers have different features, but it is common for all debuggers to be able to do the following things:

Set a breakpoint in your program. A breakpoint is a specific line in your code where you would like to stop execution of the program, so you can take a look at what's going on nearby. Step line-by-line through the program. Code only ever executes line by line, but it happens too quickly for us to figure out which lines cause mistakes. Being able to step line-by-line through your code allows you to hone in on exactly what is causing a bug in your program.
For this exercise, you will find the GDB reference card useful:
https://inst.eecs.berkeley.edu/~cs61c/resources/gdb5-refcard.pdf. GDB stands for "GNU De-Bugger."

Please run and debug the following program and answer the questions.
Tips: in the online C compiler, click the “Debug” button to debug the program. Then you are able to type gdb commands in the Debug Console.

```
#include <stdio.h>

int main (int argc, char *argv[])
{
        int i, *p, count = 0;
        p = &count;

        for (i = 0; i < 5; i++) {
                count++;
                (*p)++;
        }

        printf("count = %d, Have a nice day.\n", count);
        return 0;
}
```

ASHWIN ANAND - COMPUTER ARCHITECTURE LAB 2

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

1. [10 pts] What is the output of the program? Please explain why.

OUTPUT:

count = 10, Have a nice day.

Going through the for loop, the variable count is incremented twice every time it goes through the iteration. It is incremented once by count ++ and incremented one again as reference pointer p is incremented. Pointer p points to the address of variable count. Then the variable count is printed along with a string “count= (value of count)Have a nice day.” A new line is printed, then the program exits.

2. [5 pts] What is the gdb command to set the breakpoint in line 9?

The gdb command to set the breakpoint in line 9 is break 9 or b 9

3. [5 pts] What is the gdb command to output the value of *p?

To print out of pointer p, use gdb command print *p or p *p

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

4. [2 pts] What are the differences between the continue and step commands? How do they interact with the breakpoint you established in line 9 ?

The continue command essentially runs through the entire program till it hits a breakpoint essentially pausing the program till we execute the next command. The step command runs through the program line by line and eventually pauses when it encounters the breakpoint till we execute the next command.

Exercise 3 [20 pts]

Please debug the following program and answer the following questions.

```
#include <stdio.h>

typedef struct node {
        int value;
        struct node *next;
} node;

int ll_has_cycle(node *first) {
        node * head = first;
        while (head->next) {
              head = head->next;
                 if (head == first)
                        return 1;
        }
        return 0;

}

void test_ll_has_cycle(void) {
        int i,j;
        node nodes[5];
        for(i=0; i < sizeof(nodes)/sizeof(node); i++) {
                 nodes[i].next = NULL;
```

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

```
                nodes[i].value = i;
        }
        nodes[0].next = &nodes[1];
        nodes[1].next = &nodes[2];
        nodes[2].next = &nodes[3];

        printf("Checking first list for cycles. There should be none, ll_has_cycle says it has
%s cycle\n",
ll_has_cycle(&nodes[0])?"a":"no");
        printf("Checking length-zero list for cycles. There should be none, ll_has_cycle
              says it has %s cycle\n",
         ll_has_cycle(NULL)?"a":"no");

            printf("A node value is: %d", nodes[0].value);
}

int main(void) {
   test_ll_has_cycle();
   return 0;
}
```

1. [10 pts] What is the output of this program?

The output of the program is the following String printed on the screen:

Checking first list for cycles. There should be none, 11_has_cycle says it has no cycle

The program terminates with a Segmentation fault error, displaying exit code 139

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

2. [10 pts] Is there a bug/fault you see from the output console? If so, please explain why, fix the bug, and describe how you fix the bug.

The existent bug in the program is that it does not check for a null cycle detection. When the node f is null, any operation performed on it will give an error, hence there needs to be a check if the node f is not null, only then proceed further. The proper way to solve this bug is by adding an if statement after this line " int ll_has_cycle(node *first) " , type in the if statement as if(first ==NULL) { return 0;) . Hence now, in the fixed code when node f is null, the program smoothly exits with no error. A picture of the fixed code with new output based of bug being fixed is posted below:

```c
#include <stdio.h>
typedef struct node {
int value;
struct node *next;
} node;
int ll_has_cycle(node *first) {
    if(first==NULL){
        return 0;
    }
node * head = first;
while (head->next) {
head = head->next;
if (head == first)
return 1;
}
return 0;
}
void test_ll_has_cycle(void) {
int i,j;
node nodes[5];
for(i=0; i < sizeof(nodes)/sizeof(node); i++) {
nodes[i].next = NULL;
nodes[i].value = i;
}
nodes[0].next = &nodes[1];
nodes[1].next = &nodes[2];
nodes[2].next = &nodes[3];

printf("Checking first list for cycles. There should be none, ll_has_cycle says it has %s cycle\n",
ll_has_cycle(&nodes[0])?"a":"no");
printf("Checking length-zero list for cycles. There should be none, ll_has_cycle says it has %s cycle\n",
        ll_has_cycle(NULL)?"a":"no");

    printf("A node value is: %d", nodes[0].value);
}
int main(void) {
test_ll_has_cycle();
return 0;
}
```

```
Input
Checking first list for cycles. There should be none, ll_has_cycle says it has no cycle
Checking length-zero list for cycles. There should be none, ll_has_cycle says it has no cyc
le
A node value is: 0

...Program finished with exit code 0
Press ENTER to exit console.
```

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

Exercise 4 [15 pts]

Please run the following program and answer the questions.

```
#include <stdio.h>

int main()
{
        int a[5] = {2,5,12,4,7};
        int *arr_pointer = &a[2];
        *arr_pointer = 20;
        arr_pointer++;
        printf("%d\n", *arr_pointer);
        return 0;
}
```

For each line of the main function (lines 5 through 8 in particular), consider how the variables **a** and **arr_pointer** change as the program progresses. Explain what the values of **a** and **arr_pointer** are at each major line and why it has such value. (Hint: the gdb step command is very helpful for this problem).

Line 5:

arr_pointer = 12 is because pointer arr_pointer points to the address of a[2] = 12

Line 6:

arr_pointer = 20 as the value of a[2] gets replaced with 12 given arr_pointer still points to reference address of a[2], and the the array a is {2,5,20,4,7}

Line 7:

arr_pointer is incremented by 1 from 2 to 3 , so now pointer arr_point is referring a[3] which holds the value 4, so a[3] = 4

Line 8:

As pointer arr_pointer references a[3], value 4 is printed out

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

Exercise 5 [25 pts]

Implement a C program that reorders the elements of an array of integers such that the new order is in **ascending order** (i.e. the first number is the smallest). You must have a main function and a swap function.

- The function int main() will declare an array with the values {5,100,64,32,7,12}. This array will be passed to the swap function.
- The function void swap() will perform the necessary operations to reorder the elements of the array. Note that because swap is a void function, we cannot return the array back to the main function, meaning we must manipulate memory instead!
- After swap() is finished, have main() print the original array to show the new element order.
- Do not hard code your solution! If we test your code with different array values, we should still get the correct output.

The following link may help you get started:
https://www.tutorialspoint.com/cprogramming/c_function_call_by_reference.htm

Picture of my code is shown below and is attached to the submission of this assignment on sakai:

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

```c
//  Ashwin Anand aa2041 aa2041@scarletmail.rutgers.edu
//
// C program to accept n numbers and sort them in an ascending order
//

# include <stdio.h>

void swap(int array[],int n)
{
    int tempval=0;
    for(int a=0;a<n;a++)
    {
        for(int b=a+1;b<n;b++)
        {
            if(array[a]>array[b])
            {
                tempval = array[a];
                array[a]=array[b];
                array[b]=tempval;
            }
        }
    }
}

    int main()
{

    int numofinputs;
    printf("Enter the number of inputs: \n");
    scanf("%d", &numofinputs);
    int numbersarray [numofinputs];
    printf("Enter the numbers: \n");
    for (int i = 0; i < numofinputs; ++i)
    scanf("%d", &numbersarray[i]);
    swap(numbersarray,numofinputs);
    printf("The ascendeding array of numbers: \n");
    for (int j = 0; j < numofinputs; ++j)
    printf("%d\n", numbersarray[j]);
}
```

Input

```
Enter the number of inputs:
6
Enter the numbers:
5
100
64
23
7
12
The ascendeding array of numbers:
5
7
12
23
64
100


...Program finished with exit code 0
Press ENTER to exit console.
```