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>

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++) {
          printf("RU ");
```

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

```
     }
     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.

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!
```

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;
}
```

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.

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

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

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?

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

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;
          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");
```

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

```
    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?

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.

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).

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