Lab 2
Introduction to C Programming Language

Qadis Chaudhry

October 13, 2021

Exercise 1

```
Exercise 1:
====================
Rutgers!
Have a great semester!

Hello Scarlet Knights!
```

1. This is the output of the code above and it is based specifically on the values of the variables, $v0$, $v1$, $v2$, and $v3$. The first line is outputted unconditionally, where `\n` signals the start of a new line. The variable $v0$ is used in a switch case and since it has a value of 5, the output will be `"Rutgers!"` followed by a new line. The line after that is the default case for the switch meaning it will always be outputted. After that is a blank line which in the code is a result of the `printf("\n");` which prints a blank line. The `"Hello"` in the last line is a result of the if-else statement, where the condition is false due to $v2$ not equalling 6, meaning the else part will execute which is `s = "Hello"`. After this is another conditional if-else statement where the condition will always be false since, logically, there is no way $v3$ does not equal itself, and therefore the else section will execute. This part of the code simply prints `"Scarlet Knights!\n"` and with that is the end of the program.

2. For the `for` loop, there is an inherent flaw with how the iteration is defined, and that is the fact that if $v1$ is greater than a, the loop will run forever. Since the loop subtracts 1 from a for each iteration, and the condition is true if $v1$ is greater than a, $v1$ will always be greater than a. The condition for the loop will always be true and it will run forever. Any value of $v1$ that is greater than 5 will cause this issue since $a = 5$.

Exercise 2

count = 6, Have a wonderful day.

1. The output of the program is shown above. In this code three variables are defined, `i, p` and `count.` Here, the variable `p` is set to be pointer to `count` and this means that changing the value of `*p` will change the value of `count`. When we reach the `for` loop, this concept is important since both `*p` and `count` are being changed. This means at every iteration the value of `count` changes twice leaving the final value being 6 over the three iterations.

2. The command to set a breakpoint at a line is simply `break` and then the name of the file and the line number. To specify a breakpoint at line 7 in the file `main.c`, the command would be `break main.c:7`.

3. The `break... if` *expr* command might be useful for when dealing with loops and we need to break conditionally at a value of a specific variable. There can be cases where a loop runs and there is a bug within the loop causing some variable to manifest a value that it is not supposed to. If for example we know the desired value, we can have the condition be `if not(desired value)` and therefore the code will stop at the point where the variable takes on the erroneous value.

Exercise 3

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

1. The output of the program initially is a bunch of errors referring to the `printf` statements. After the bugs are fixed, this is the intended output of the program, shown above.

2. The bug/fault of this program was the format of the `printf` statements. Since they spanned two lines, there was no quotation mark to end the argument of the printed statement. To fix the program, we simply needed to add a set of two quotations, one at the end of the first line where it splits and another where the second line begins. Another fix is to make the print argument only span one line and the code will run properly.

Exercise 4

For this program we have two variables, `a` and `arr_pointer`, where `a` is an array holding 5 values and `arr_pointer` is set to point to the third index, or the forth value, within the array. For the time right after these two are assigned, the value of `a` is `{42, 31, 65, 78, 9}` and since `arr_pointer` point to the third index of the array, it holds the value `78` initially. Right after these first two lines of the `main` function, `*arr_pointer` is set equal to `20`, and since this is a pointer to `a[3]`, the value of `a[3]` is set to `20`. This makes the new value of `a`, `{42, 31, 65, 20, 9}`. The line after this sets the value of the point to itself plus 1 meaning the new value of `*arr_pointer = a[4]`. Lastly, the program outputs the decimal value of `*arr_pointer`, the value stored in `a[4]`, which is `9`.

Exercise 5

C code:

```
#include <stdio.h>

void swap(int *x, int *y) {
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    return;
}

int main() {
    int arr[6] = {12, 100, 10, 1000, 11, 12};
    int size = sizeof arr / 4;
    int *p1 = &arr[0];
    int *p2 = &arr[size-1];
    printf("Orignial Array: \n");
    for (int i=0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    for (int i=0; i < size / 2; i++) {
        swap(p1, p2);
        p1++;
        p2--;
    }
    printf("Reversed Array: \n");
    for (int i=0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Output:

```
Orignial Array:
12 100 10 1000 11 12
Reversed Array:
12 11 1000 10 100 12
```