Mohammad Zoraiz

Question #1

a. **Convert +57$_{10}$ to 8-bit 2s complement integer representation in binary and hexadecimal. You must show your work!**
   i. Work:
      1. Step 1 (Convert 57 to 6-Bit Binary): 111001
      2. Step 2 (Make it signed) - Reached Signed Binary: 00 111001
      3. Step 3 (Convert to Hexadecimal):
         a. Last 4 Binary Values: 1001 = 9
         b. First 4 Binary Values: 0011 = 3
   ii. Signed 8-Bit Binary value of +57$_{10}$ = ***00111001***
   iii. Hexadecimal value of +57$_{10}$ = ***00x39***

b. **Convert -24$_{10}$ to 8-bit 2s complement integer representation in binary and hexadecimal. You must show your work!**
   i. Work:
      1. Step 1 (Convert 24 to 6-Bit Binary): 011000
      2. Step 2 (Make it signed): 00 011000
      3. Step 3 (Invert the bits): 11100111
      4. Step 4 (Add 1 to the result) - Reached Signed Binary: 11101000
      5. Step 5 (Convert to Hexadecimal):
         a. Last 4 Binary Values: 1000 = E
         b. Last 4 Binary Values: 1110 = 8
   ii. Signed 8-Bit Binary value of -24$_{10}$: ***11101000***
   iii. Hexadecimal value of -24$_{10}$: ***00xE8***

c. **Convert +57.0$_{10}$ to 32-bit IEEE floating point representation in binary and hexadecimal. You must show your work!**
   i. Work 1:
      1. Step 1 (Convert 57 to binary): 111001, fractional part is 0
      2. Step 2 (Normalize the binary representation): 1.11001×2^5
      3. Step 3 (Sign): Sign is Positive - 0
      4. Step 4 (Exponent$_{10}$): 127+5 = 132
         a. Step 4b (132 in Binary): 10000100
      5. Step 5 (Combining):
         a. Binary Representation: **0 10000100 11001000000000000000000**
      6. Step 6 (Converting Hexadecimal):
         a. Hexadecimal Representation: **0x42640000**
            i. 0100 - 4
            ii. 0010 - 2
            iii. 0110 - 6
            iv. 0100 - 4
            v. 0000 - 0
            vi. 0000 - 0
            vii. 0000 - 0
            viii. 0000 - 0

d. **Convert -1.75_{10} to 32-bit IEEE floating point representation in binary and hexadecimal. You must show your work!**
   i. Step 1 (Convert 1 to binary): 1.11
   ii. Step 2 (Normalize the binary representation): 1.11×2^0
   iii. Step 3 (Sign): Sign is Negative - 1
   iv. Step 4 (Exponent_{10}): 127+0 = 127
      1. Step 4b (127 in Binary): 0111 1111
   v. Step 5 (Combining):
      1. Binary Representation: **1 01111111 11000000000000000000000**
   vi. Step 6 (Converting Hexadecimal):
      1. Hexadecimal Representation: **0xBFE00000**
         a. 1011 - B
         b. 1111 - F
         c. 1110 - E
         d. 0000 - 0
         e. 0000 - 0
         f. 0000 - 0
         g. 0000 - 0
         h. 0000 - 0

e. **Represent the ASCII string "Spring 24" (not including the quotes) in hexadecimal.**
   i. Hexadecimal Conversions: S: 53, p: 70, r: 72, i: 69, n: 6E, g: 67, (space): 20, 2: 32, 4: 34
   ii. 53 70 72 69 6E 67 20 32 34

f. **Give an example of an integer that cannot be represented as a 32-bit signed integer.**
   i. An integer that cannot be represented as a 32-bit signed integer are any integers less than $-1\text{x}(2^{32})$ or greater than 2^{32}, an example of an integer that cannot be represented 2,500,000,000.

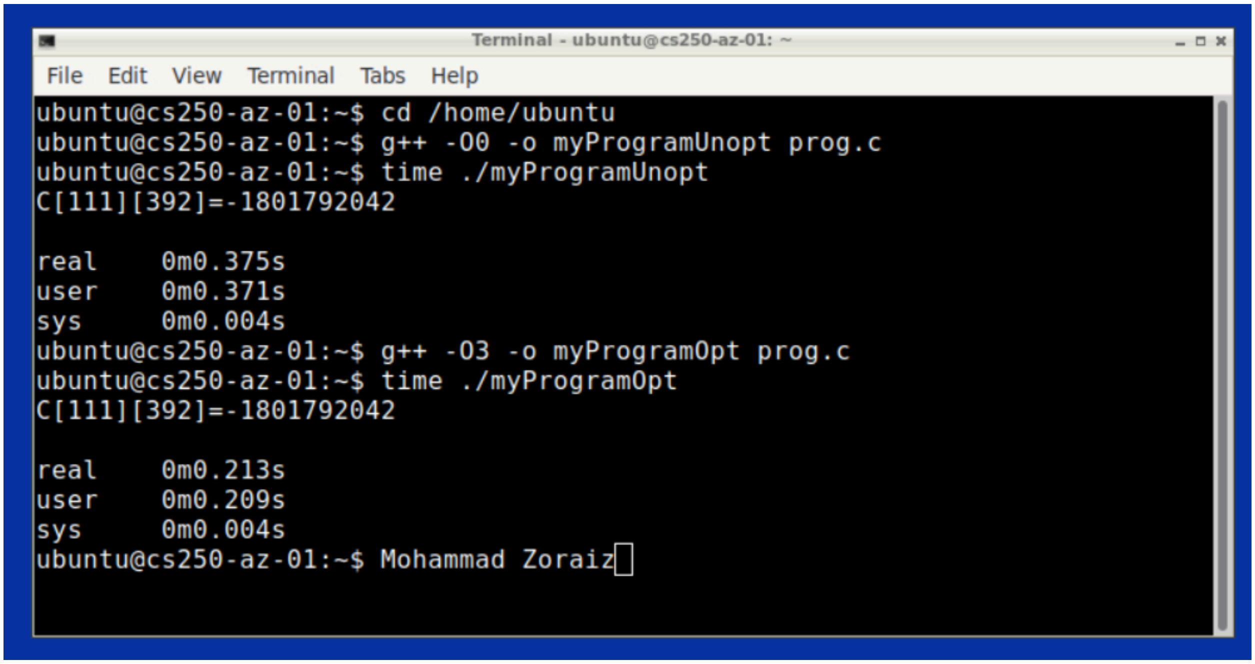
Question #2

a. **Where do each of the following variables live (global data, stack, or heap)?**
   i. **a**
      1. Global data, defined outside of function.
   ii. **b_ptr**
      1. On stack, local variable in the main function.
   iii. ***b_ptr**
      1. On heap, value stored at the address of the pointer b_ptr.
   iv. **e_ptr**
      1. On stack, local variable in the main function storing address of e.
   v. ***e_ptr**
      1. Global data, refers to the address stored in e_ptr, a global variable.

b. **What is the value returned by main()?**
   i. The main function returns 10. This is since, effectively, *e_ptr points to a which is 10.2, and *(b_ptr)+1 is 7. Moreover, foo() causes *x to be 10.2, *y to be 4.0, and

z to be 10.2. As *x > *y, foo() returns z as 10.2. Finally, within the if condition, since c>10.0, it returns c, however, is implicitly converted to an int, which is 10.

Question #3

a. 

b. The time required to compile the optimized prog.c is approximately only ⅔ of the time required to run the unoptimized version of prog.c. Evidently, it shows that the -O3 flag in comparison to the -O0 flag helps for the program to perform more efficiently and optimized by a significant factor.

Mohammad Zoraiz

Question 4a: Tribonacci

C tribonacci.c > main(int, char * [])

```c
//Mohammad Zoraiz

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int counter;

    if(sscanf(argv[1], "%d", &counter) != 1)
    {
        return EXIT_FAILURE;
    }

    if (counter <= 0)
    {
        return EXIT_FAILURE;
    }

    int first = 1;
    int second = 1;
    int third = 2;
    int total = 3;

    if (counter >= 1) printf("%d\n", first);
    if (counter >= 2) printf("%d\n", second);
    if (counter >= 3) printf("%d\n", third);

    for (int x = 4; x <= counter; x++)
    {
        total = first + second + third;
        printf("%d\n", total);

        first = second;
        second = third;
        third = total;
    }

    return EXIT_SUCCESS;
}
```

```
ubuntu@cs250-az-01:~/homework-3-c$ g++ -o -o tribonacci tribonacci.c
ubuntu@cs250-az-01:~/homework-3-c$ ./hwtest.py tribonacci
Running tests for tribonacci...
Test 0     n = 1                                          Pass
Test 1     n = 2                                          Pass
Test 2     n = 4                                          Pass
Test 3     n = 7                                          Pass
Test 4     n = 10                                         Pass
Done running tests for tribonacci.
```

Mohammad Zoraiz

Question 4b: recurse.c

recurse.c

homework-3-c > recurse.c > main(int, char * [])

```c
//Mohammad Zoraiz - ECE250

#include <stdio.h>
#include <stdlib.h>

int recurse(int multiplier)
{

    if (multiplier == 0)
    {
        return 2;
    }
    else
    {
        return (3*multiplier)-2*recurse(multiplier-1)+7;
    }

}

int main(int argc, char *argv[])
{

    if (argc != 2)
    {
        return EXIT_FAILURE;
    }

    int multiplier;

    sscanf(argv[1], "%d", &multiplier);

    if (multiplier < 0)
    {
        return EXIT_FAILURE;
    }

    int final = recurse(multiplier);

    printf("%d\n", final);

    return EXIT_SUCCESS;
}
```

```
mzoraiz@MZoraizP15:~/CS250HW3$ cd homework-3-c
mzoraiz@MZoraizP15:~/CS250HW3/homework-3-c$ g++ -g -o recurse recurse.c
mzoraiz@MZoraizP15:~/CS250HW3/homework-3-c$ ./hwtest.py recurse
Running tests for recurse...
Test 0      n = 0                                          Pass
Test 1      n = 2                                          Pass
Test 2      n = 4                                          Pass
Test 3      n = 7                                          Pass
Done running tests for recurse.
```