Reta Gela
28 September 2023

ENEB 340 Lab 5

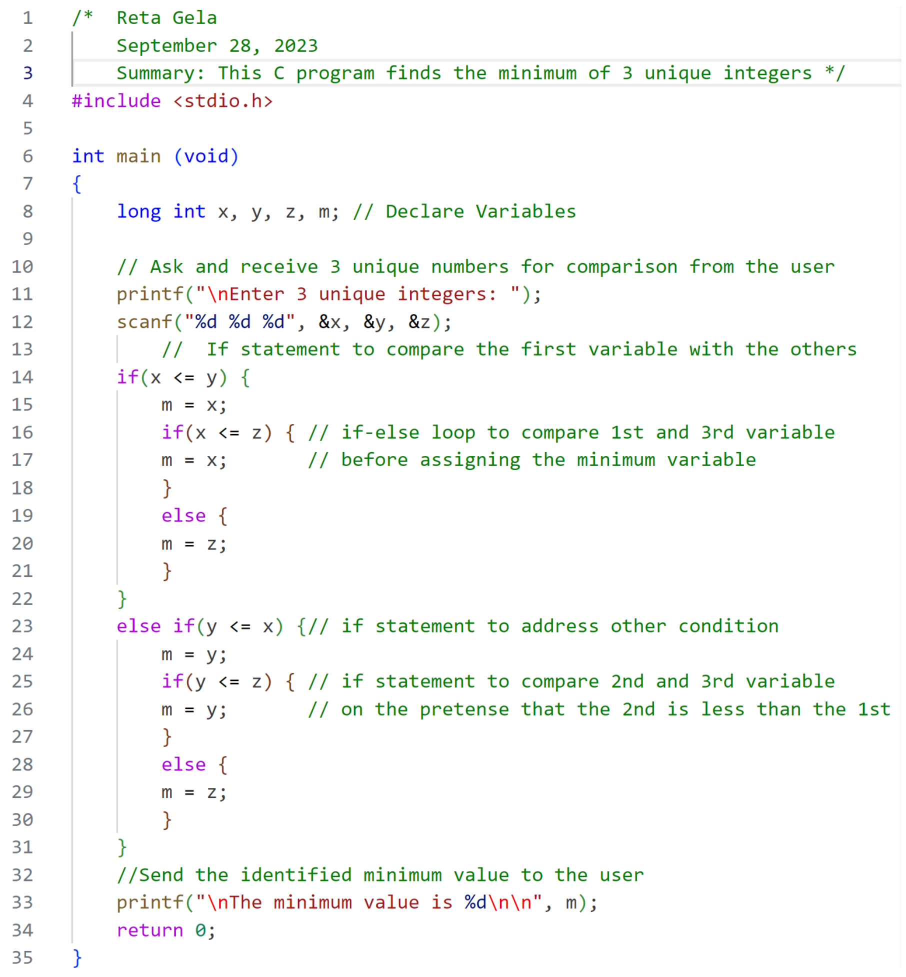
Introduction

This lab investigation involved creating 4 different C programs using things like nested loops and bitwise operators. These 4 programs were created, implemented and tested using the Raspberry Pi OS.

Procedure

Part 1: Minimum of 3 Program

In the first program, I wrote a program that displayed the value of the minimum of three integers. This was created using several if-else statements to compare the 3 input values. The general structure of the program used several nested if statements to systematically compare the three integers using just two of the integers. The structure of the program is below.

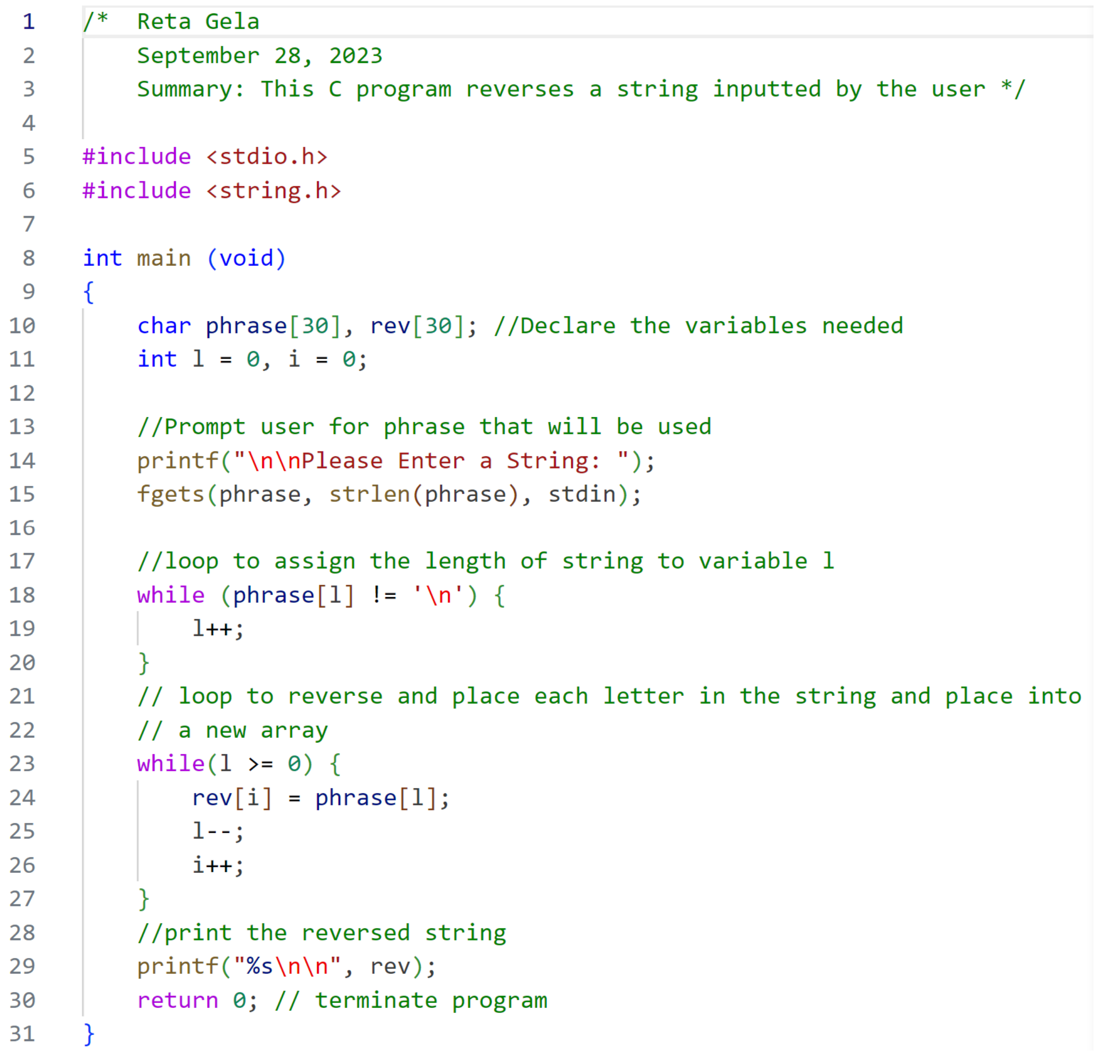


```c
/*  Reta Gela
    September 28, 2023
    Summary: This C program finds the minimum of 3 unique integers */
#include <stdio.h>

int main (void)
{
    long int x, y, z, m; // Declare Variables

    // Ask and receive 3 unique numbers for comparison from the user
    printf("\nEnter 3 unique integers: ");
    scanf("%d %d %d", &x, &y, &z);
        //  If statement to compare the first variable with the others
    if(x <= y) {
        m = x;
        if(x <= z) { // if-else loop to compare 1st and 3rd variable
        m = x;       // before assigning the minimum variable
        }
        else {
        m = z;
        }
    }
    else if(y <= x) {// if statement to address other condition
        m = y;
        if(y <= z) { // if statement to compare 2nd and 3rd variable
        m = y;       // on the pretense that the 2nd is less than the 1st
        }
        else {
        m = z;
        }
    }
    //Send the identified minimum value to the user
    printf("\nThe minimum value is %d\n\n", m);
    return 0;
}
```

Figure 1: Minimum of 3 Integers

Program II: String Reverse

In this assignment, I created a program that received a string from the user and outputted the string in reverse. This program used two while loops and 2 register variables and 2 array strings to implement this objective. The pseudocode and final code are pictured in Figure 2.



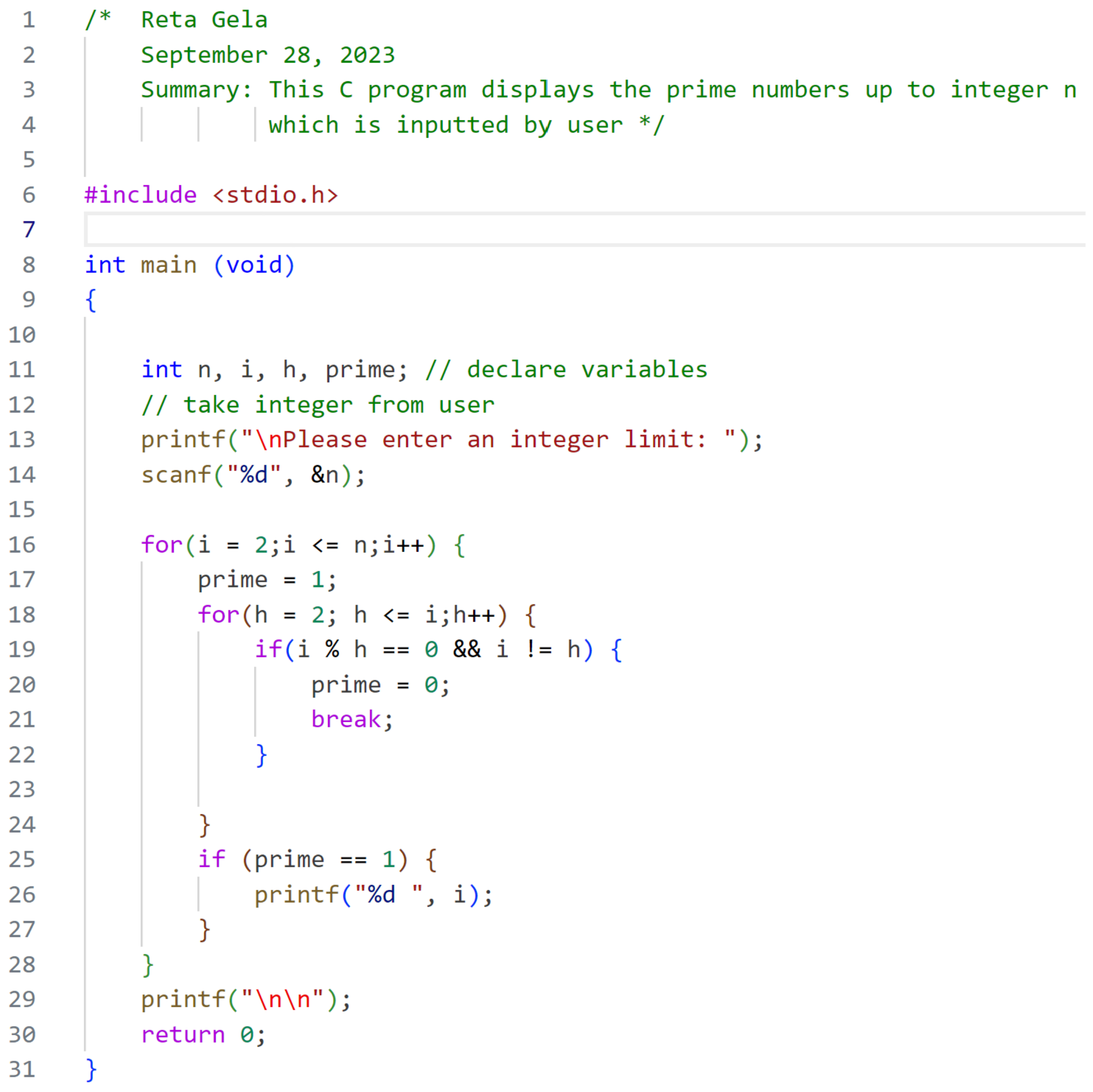
```c
/*  Reta Gela
    September 28, 2023
    Summary: This C program reverses a string inputted by the user */

#include <stdio.h>
#include <string.h>

int main (void)
{
    char phrase[30], rev[30]; //Declare the variables needed
    int l = 0, i = 0;

    //Prompt user for phrase that will be used
    printf("\n\nPlease Enter a String: ");
    fgets(phrase, strlen(phrase), stdin);

    //loop to assign the length of string to variable l
    while (phrase[l] != '\n') {
        l++;
    }
    // loop to reverse and place each letter in the string and place into
    // a new array
    while(l >= 0) {
        rev[i] = phrase[l];
        l--;
        i++;
    }
    //print the reversed string
    printf("%s\n\n", rev);
    return 0; // terminate program
}
```

Figure 2: String Reverse Source Code

In the program above, a string is taken from the user and placed into an array using the function ‘fgets().’ This function allows for an entire string to be stored including space characters. This will allow the reversed string to not be cut short. Then the program enters a while loop, to assign the length of the string to variable ‘l.’ This variable ‘l’ is then used as the conditional operand for which the reverse string to be generated by assigning the last bit of the string to the first bit of the other array declared, ‘rev[30].’ The register variable ‘i’ is incremented by 1 and the ‘l’ variable is decremented by 1. This while loop only stops when the ‘l’ variable is less than 0. This is because the first bit of a string is numbered as the ‘0th’ bit or ‘string[0].’ The reversed string is then printed and the program terminates.

Program III: Prime Number Generator

This program's main function is to take an integer from the user and produce all the prime numbers from 1 to that integer. I had to use 2 nested for loops and several register variables in order to test all numbers 1 by 1 until I reached the integer given. The pseudocode comments are below along with the code.



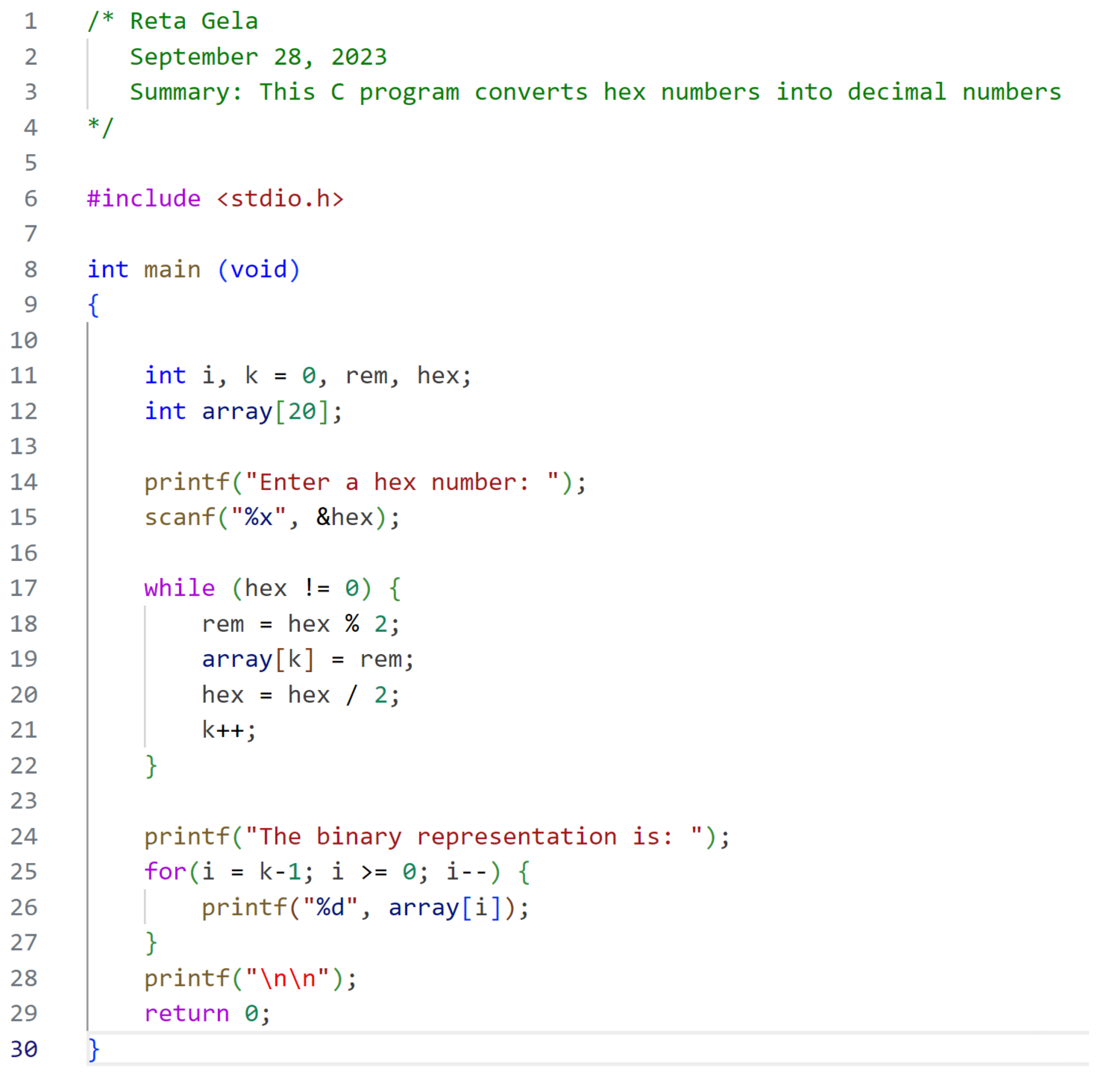
```c
/*  Reta Gela
    September 28, 2023
    Summary: This C program displays the prime numbers up to integer n
             which is inputted by user */

#include <stdio.h>

int main (void)
{

    int n, i, h, prime; // declare variables
    // take integer from user
    printf("\nPlease enter an integer limit: ");
    scanf("%d", &n);

    for(i = 2;i <= n;i++) {
        prime = 1;
        for(h = 2; h <= i;h++) {
            if(i % h == 0 && i != h) {
                prime = 0;
                break;
            }

        }
        if (prime == 1) {
            printf("%d ", i);
        }
    }
    printf("\n\n");
    return 0;
}
```

Figure 3: Prime Number List Generator Source Code

In the program above, several register variables are first declared and the user is prompted to enter an integer limit. Then, two nested for loops are used to first regulate that the integer limit is not exceeded, and then sets the register variable 'prime' to be the condition of whether or not an integer is printed in the resulting list. This is done by integrating an if statement that skips an integer during an iteration if it is divisible by another number that it is not itself and 1. The and operator '&&' and "!=" are important here. Thus, the resulting string is printed 1 by 1 until the integer limit is reached.

Program IV: Hex to Binary Converter

For this assignment, I created a program that takes a single byte in hex and converts it to its binary equivalent. I used several register variables and two loops to change the hex value bit by bit. The code I synthesized is in Figure 4.



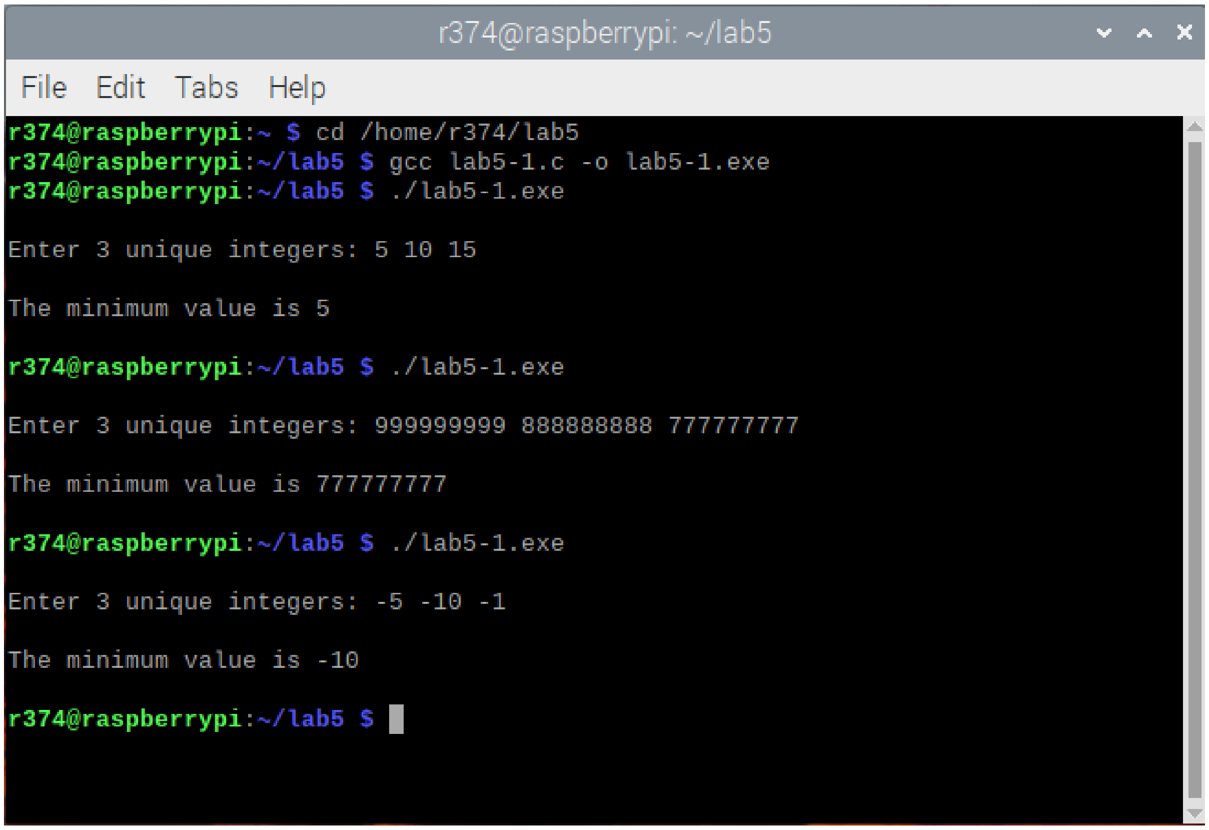
```c
/* Reta Gela
   September 28, 2023
   Summary: This C program converts hex numbers into decimal numbers
*/

#include <stdio.h>

int main (void)
{

    int i, k = 0, rem, hex;
    int array[20];

    printf("Enter a hex number: ");
    scanf("%x", &hex);

    while (hex != 0) {
        rem = hex % 2;
        array[k] = rem;
        hex = hex / 2;
        k++;
    }

    printf("The binary representation is: ");
    for(i = k-1; i >= 0; i--) {
        printf("%d", array[i]);
    }
    printf("\n\n");
    return 0;
}
```

Figure 4: Hex to Binary Converter

The code starts off by prompting the user for the hex number and places it into the variable ‘hex.’ Then, it enters a loop whereby the hex number is modulated by the binary base number 2 using the modulus operator. The result is then placed into the first element of an array. The hex value is set to the quotient of itself divided by 2 and the register variable k is incremented by 1. The loop stops when the hex value equals zero after the repeated division. Finally, the array is displayed 1 by 1 using a for loop and register variable ‘i’ that starts at the last digit of the array and displays the digit 1 by 1 in reverse. This is because the original array is reversed because algebraically, the binary equivalent is written from right to left after successive division. But in this program, it is written from left to right and then reversed for ease of coding.

Test Case Results

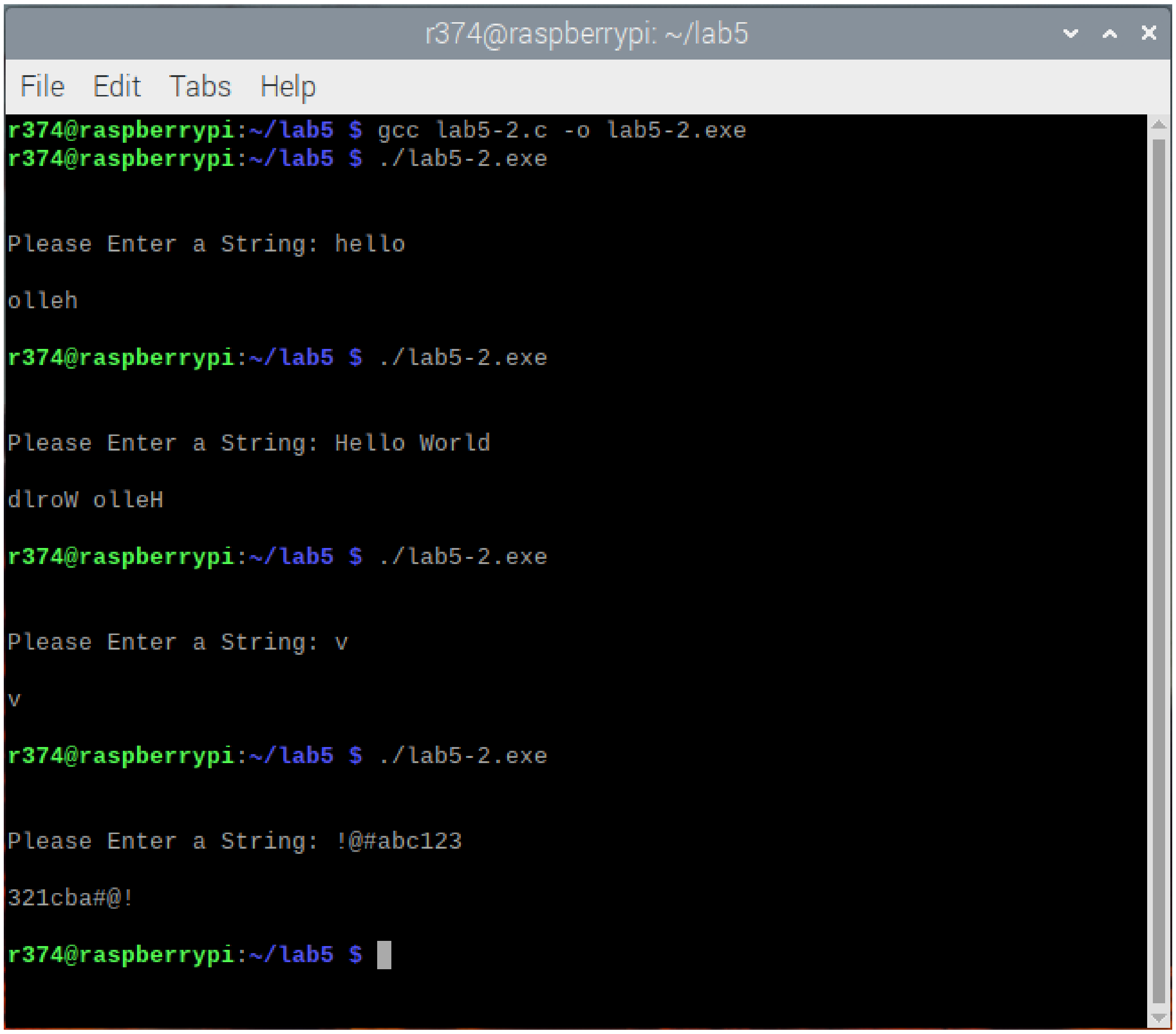
<u>Minimum of 3 Program</u>



```
r374@raspberrypi:~ $ cd /home/r374/lab5
r374@raspberrypi:~/lab5 $ gcc lab5-1.c -o lab5-1.exe
r374@raspberrypi:~/lab5 $ ./lab5-1.exe

Enter 3 unique integers: 5 10 15

The minimum value is 5

r374@raspberrypi:~/lab5 $ ./lab5-1.exe

Enter 3 unique integers: 999999999 888888888 777777777

The minimum value is 777777777

r374@raspberrypi:~/lab5 $ ./lab5-1.exe

Enter 3 unique integers: -5 -10 -1

The minimum value is -10

r374@raspberrypi:~/lab5 $ 
```

Figure 5: Test Case Results from Raspberry Pi Terminal

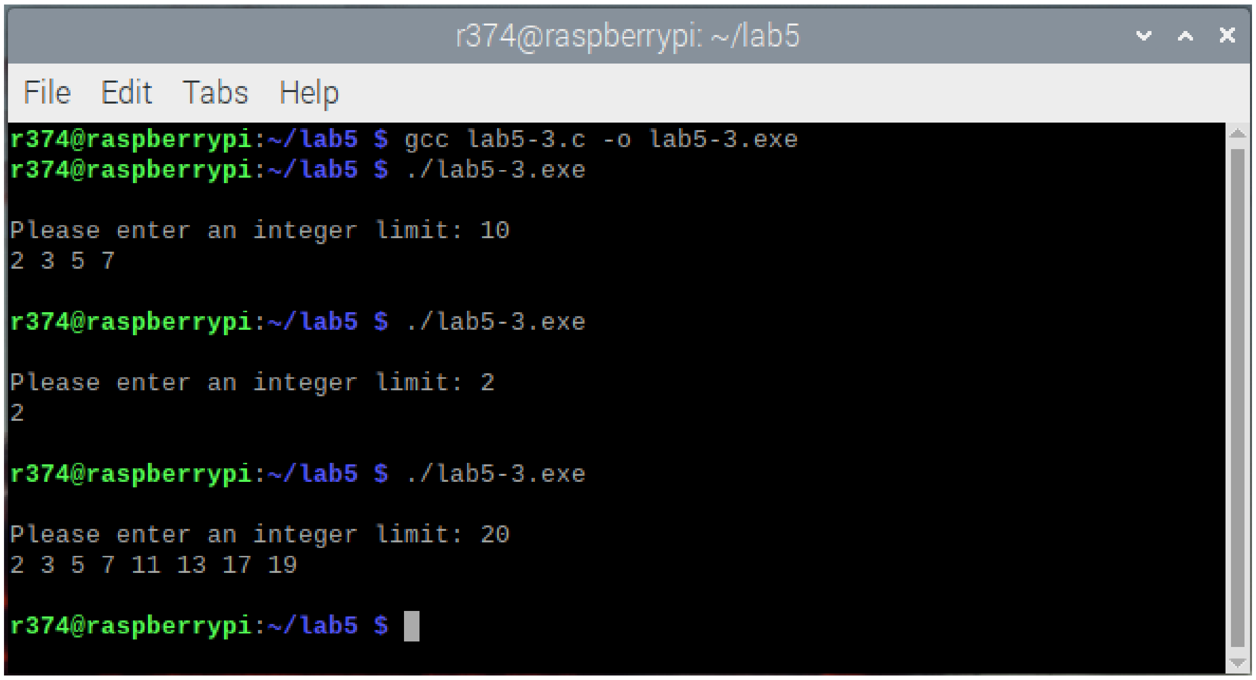
String Reverse Program Test Case Results



```
r374@raspberrypi:~/lab5 $ gcc lab5-2.c -o lab5-2.exe
r374@raspberrypi:~/lab5 $ ./lab5-2.exe


Please Enter a String: hello

olleh

r374@raspberrypi:~/lab5 $ ./lab5-2.exe


Please Enter a String: Hello World

dlroW olleH

r374@raspberrypi:~/lab5 $ ./lab5-2.exe


Please Enter a String: v

v

r374@raspberrypi:~/lab5 $ ./lab5-2.exe


Please Enter a String: !@#abc123

321cba#@!

r374@raspberrypi:~/lab5 $
```

Figure 6: Test Case Results from Raspberry Pi Terminal

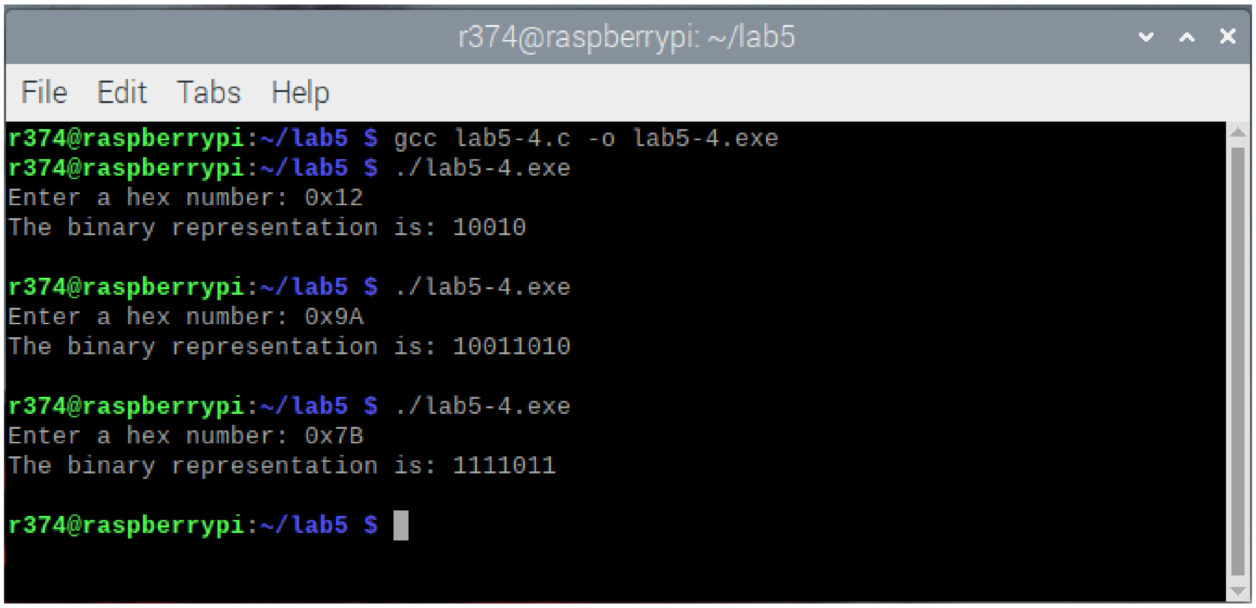
Prime Number List Generator Test Case Results



```
r374@raspberrypi:~/lab5 $ gcc lab5-3.c -o lab5-3.exe
r374@raspberrypi:~/lab5 $ ./lab5-3.exe

Please enter an integer limit: 10
2 3 5 7

r374@raspberrypi:~/lab5 $ ./lab5-3.exe

Please enter an integer limit: 2
2

r374@raspberrypi:~/lab5 $ ./lab5-3.exe

Please enter an integer limit: 20
2 3 5 7 11 13 17 19

r374@raspberrypi:~/lab5 $
```

Figure 7: Test Case Results from Raspberry Pi Terminal

Hex to Binary Converter Test Case Results



```
r374@raspberrypi:~/lab5 $ gcc lab5-4.c -o lab5-4.exe
r374@raspberrypi:~/lab5 $ ./lab5-4.exe
Enter a hex number: 0x12
The binary representation is: 10010

r374@raspberrypi:~/lab5 $ ./lab5-4.exe
Enter a hex number: 0x9A
The binary representation is: 10011010

r374@raspberrypi:~/lab5 $ ./lab5-4.exe
Enter a hex number: 0x7B
The binary representation is: 1111011

r374@raspberrypi:~/lab5 $
```

Figure 8: Test Case Results from Raspberry Pi Terminal

Conclusion

All in all, this was a successful lab investigation that expanded my knowledge of C programming concepts and syntax. I learned how to implement nested loops and if-else statements together to generate lists, nested loops for conversion programming and register variables for reversing strings. Finally, I learned other possible errors that could occur when programming like the ‘segmentation fault’ error when I ran the second program (string error). I fixed this by rearranging the limit condition operand and using the ‘fgets()’ function.

In the future, I would like to try different conversion programs and grade ordering programs.