Reta Gela
26 October 2023

ENEB 340 Lab 6 Report

Procedure

1. I plugged in and turned on the Raspberry Pi board before connecting it to the internet, keyboard, mouse and computer monitor.

<u>Part 1</u>: *Sum of Natural Numbers Program*

1. The objective for this task was to create a program that prompted and took an integer limit value from the user and calculated the sum of all natural numbers up to and including that integer limit before transmitting that back to the user.
2. I created a pseudocode within Geany before beginning to write out the final code (see Figure 1).
3. I compiled and executed the program and used the provided test cases to check the correctness of its output (see Figure 2).

<u>Part 2</u>: *Vowel & Consonant String Counter*

1. For this assignment, I created a program that takes a string from the user and copies the vowels and consonants into separate arrays in the order they were written (left to right) and prints out these arrays to the user.
2. I created the pseudocode using 2 separate functions for each component to calculate the two operations and passed the string to both consecutively. A detailed explanation is under Figure 4.
3. The results of the test cases for the program are in Figure 5.

<u>Part 3</u>: *Days in a Month Calculator*

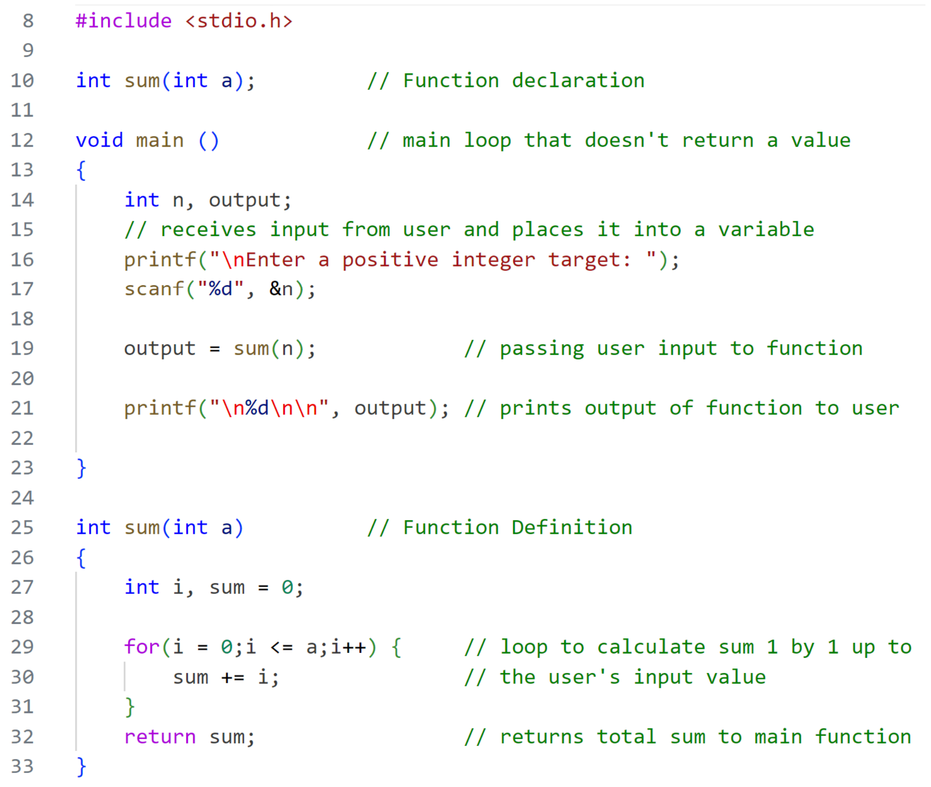
1. For this program, the goal was to create a machine that would take the specific month of a specific year and display its number of days.
2. I used a series of if else statements to construct the main derivation program and used the same function calling techniques learned in class. The detailed code is in Figure 7.
3. The output of the program for the test cases provided are in Figure 8.

<u>Part 4</u>: *Fibonacci Function*

1. This program takes the number of desired terms from the user and displays the fibonacci sequence up to that inputted term number.
2. A lengthy pseudo code was written in Geany before the final code (in Figure 9) was compiled and executed.
3. The output of the test cases are in Figure 10.

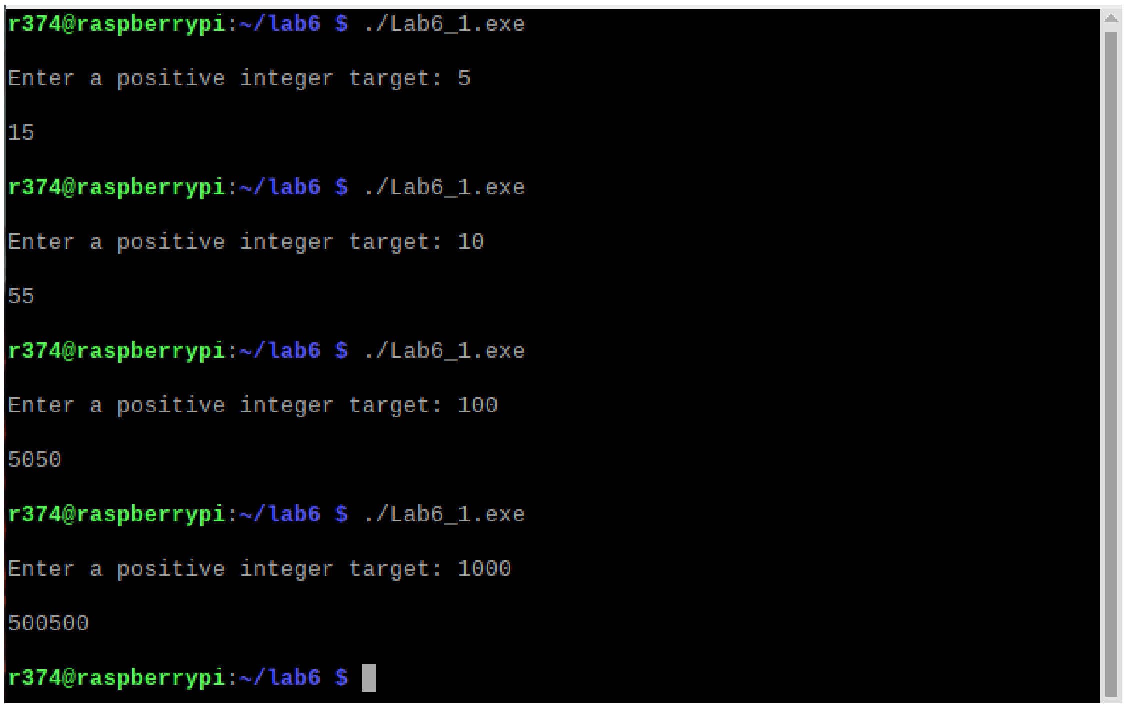
Results

<u>Program 1</u>: 'N Sum' Program



```c
#include <stdio.h>

int sum(int a);          // Function declaration

void main ()             // main loop that doesn't return a value
{
    int n, output;
    // receives input from user and places it into a variable
    printf("\nEnter a positive integer target: ");
    scanf("%d", &n);

    output = sum(n);             // passing user input to function

    printf("\n%d\n\n", output); // prints output of function to user

}

int sum(int a)           // Function Definition
{
    int i, sum = 0;

    for(i = 0;i <= a;i++) {     // loop to calculate sum 1 by 1 up to
        sum += i;               // the user's input value
    }
    return sum;                 // returns total sum to main function
}
```

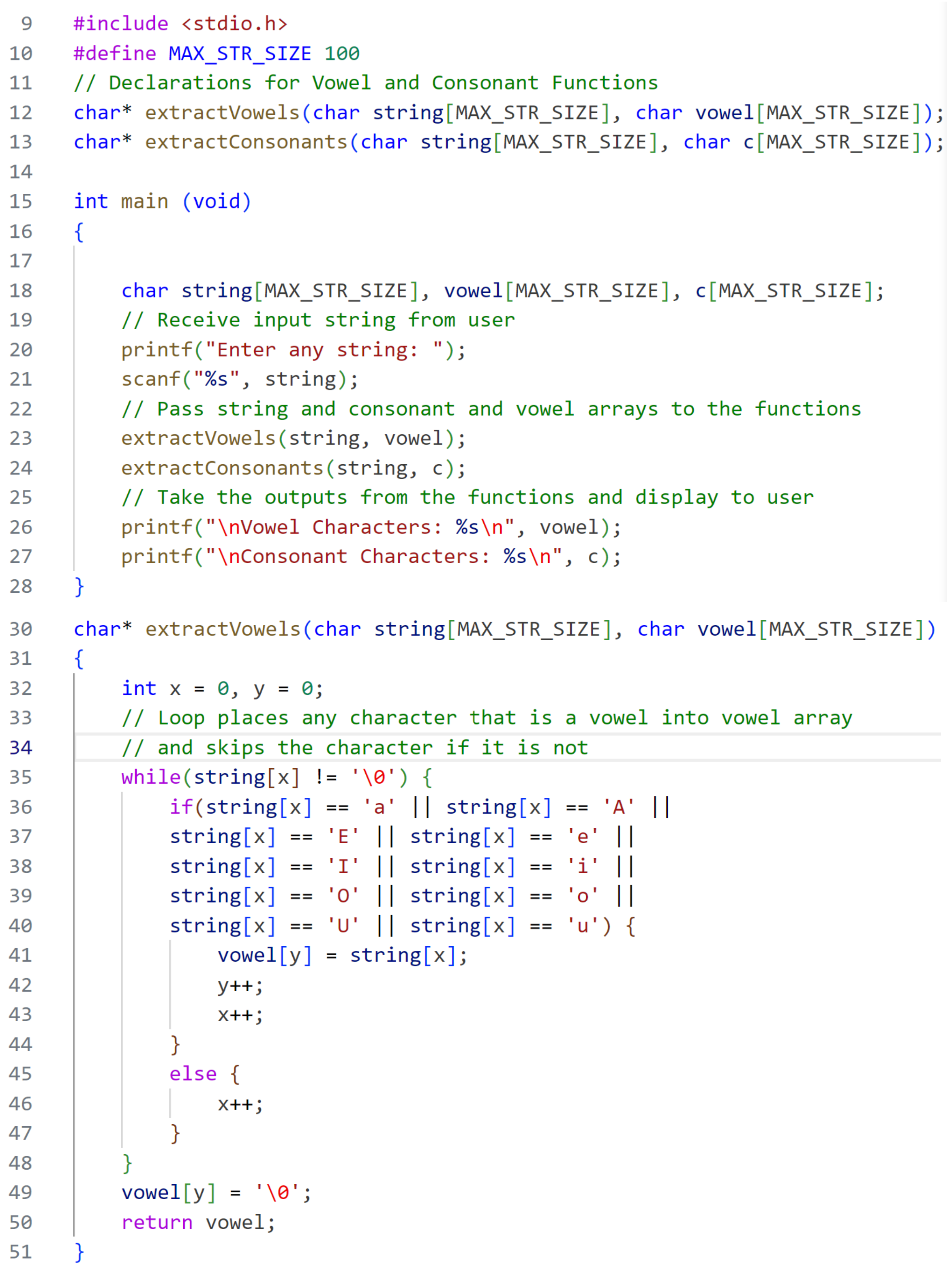
Figure 1: 'N Sum' Program Code



```
r374@raspberrypi:~/lab6 $ ./Lab6_1.exe

Enter a positive integer target: 5

15

r374@raspberrypi:~/lab6 $ ./Lab6_1.exe

Enter a positive integer target: 10

55

r374@raspberrypi:~/lab6 $ ./Lab6_1.exe

Enter a positive integer target: 100

5050

r374@raspberrypi:~/lab6 $ ./Lab6_1.exe

Enter a positive integer target: 1000

500500

r374@raspberrypi:~/lab6 $
```

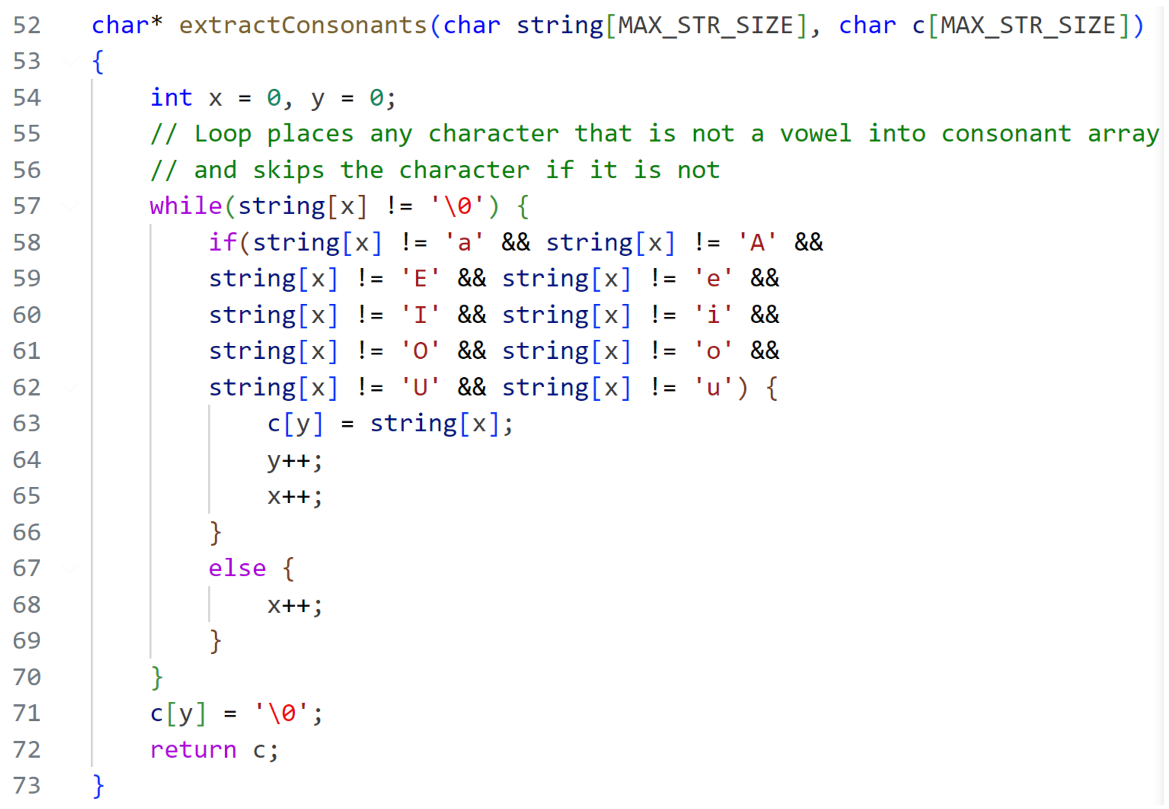
Figure 2: Test Case Outputs for 'N Sum' Program

<u>Program 2</u>: Vowel and Consonants



```c
#include <stdio.h>
#define MAX_STR_SIZE 100
// Declarations for Vowel and Consonant Functions
char* extractVowels(char string[MAX_STR_SIZE], char vowel[MAX_STR_SIZE]);
char* extractConsonants(char string[MAX_STR_SIZE], char c[MAX_STR_SIZE]);

int main (void)
{

    char string[MAX_STR_SIZE], vowel[MAX_STR_SIZE], c[MAX_STR_SIZE];
    // Receive input string from user
    printf("Enter any string: ");
    scanf("%s", string);
    // Pass string and consonant and vowel arrays to the functions
    extractVowels(string, vowel);
    extractConsonants(string, c);
    // Take the outputs from the functions and display to user
    printf("\nVowel Characters: %s\n", vowel);
    printf("\nConsonant Characters: %s\n", c);
}
char* extractVowels(char string[MAX_STR_SIZE], char vowel[MAX_STR_SIZE])
{
    int x = 0, y = 0;
    // Loop places any character that is a vowel into vowel array
    // and skips the character if it is not
    while(string[x] != '\0') {
        if(string[x] == 'a' || string[x] == 'A' ||
        string[x] == 'E' || string[x] == 'e' ||
        string[x] == 'I' || string[x] == 'i' ||
        string[x] == 'O' || string[x] == 'o' ||
        string[x] == 'U' || string[x] == 'u') {
            vowel[y] = string[x];
            y++;
            x++;
        }
        else {
            x++;
        }
    }
    vowel[y] = '\0';
    return vowel;
}
```

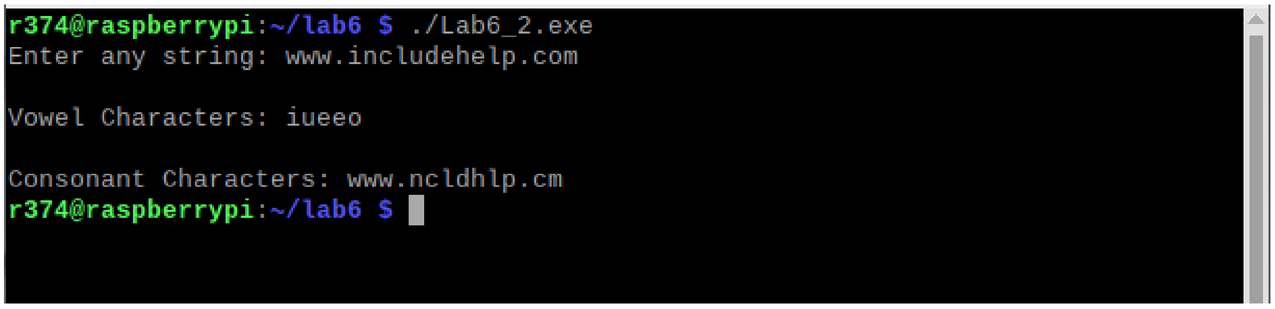
Figure 3: Vowel and Consonant Program Code (Main Loop & Vowel Function)



```c
char* extractConsonants(char string[MAX_STR_SIZE], char c[MAX_STR_SIZE])
{
    int x = 0, y = 0;
    // Loop places any character that is not a vowel into consonant array
    // and skips the character if it is not
    while(string[x] != '\0') {
        if(string[x] != 'a' && string[x] != 'A' &&
        string[x] != 'E' && string[x] != 'e' &&
        string[x] != 'I' && string[x] != 'i' &&
        string[x] != 'O' && string[x] != 'o' &&
        string[x] != 'U' && string[x] != 'u') {
            c[y] = string[x];
            y++;
            x++;
        }
        else {
            x++;
        }
    }
    c[y] = '\0';
    return c;
}
```

Figure 4: Vowel & Consonant Program (Consonant Function)

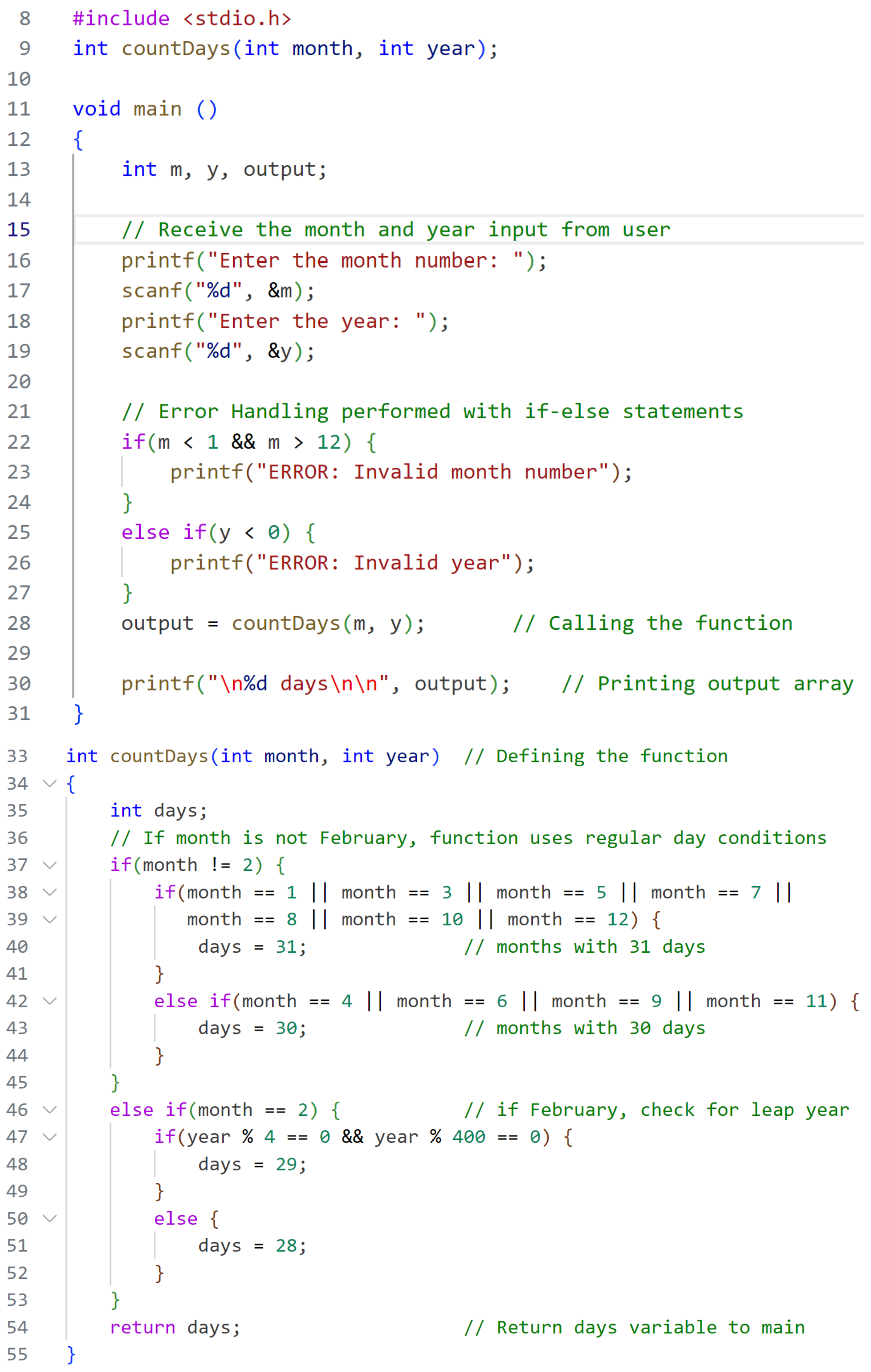
In the concatenated pictures above, the heart of the code involves creating a function that takes the inserted term and an empty string and calculates each term consecutively, placing each new value one by one into the new string before sending the final string back to the main loop, where, each term is printed out one by one from the string in order to insert a space in between each of them using the printf function.



```
r374@raspberrypi:~/lab6 $ ./Lab6_2.exe
Enter any string: www.includehelp.com

Vowel Characters: iueeo

Consonant Characters: www.ncldhlp.cm
r374@raspberrypi:~/lab6 $ 
```

Figure 5: Test Cases for Vowel & Consonant Program

<u>Program 3</u>: Month Days Calculator

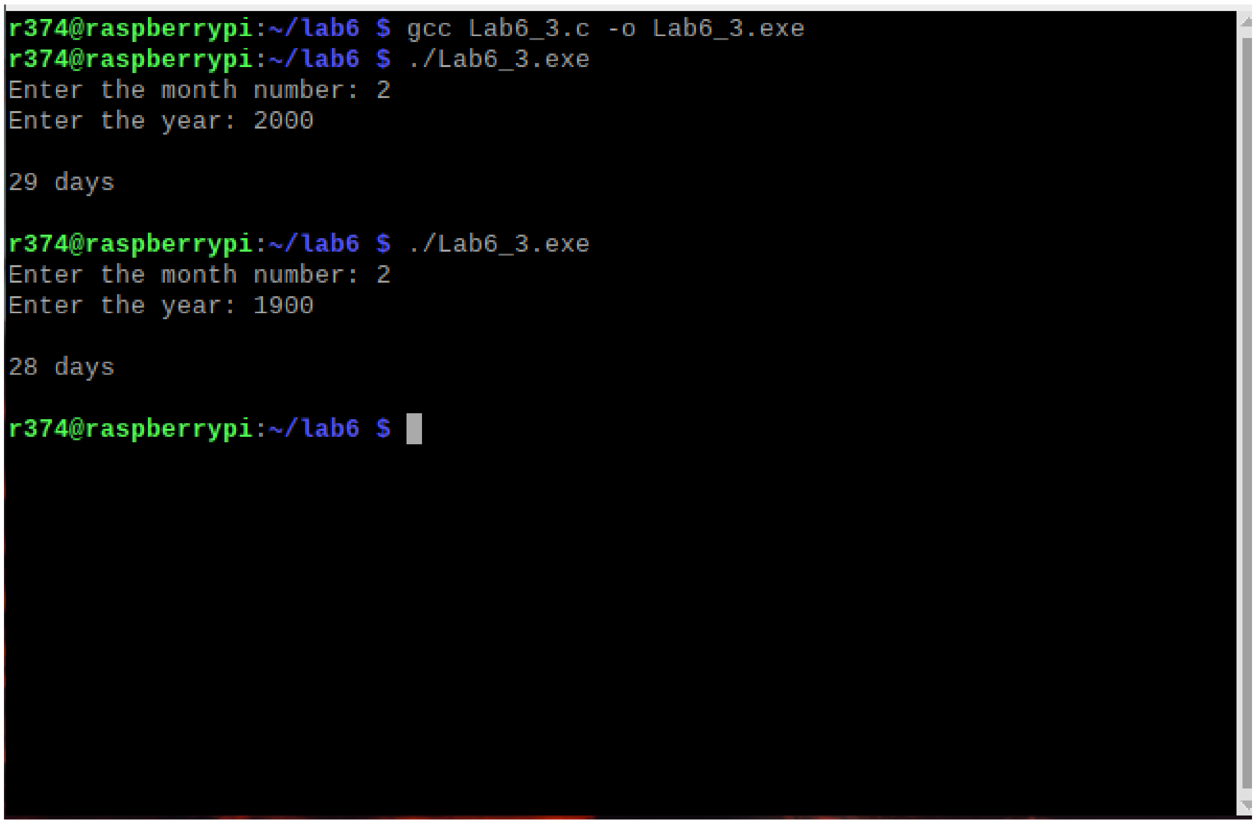


```c
#include <stdio.h>
int countDays(int month, int year);

void main ()
{
    int m, y, output;

    // Receive the month and year input from user
    printf("Enter the month number: ");
    scanf("%d", &m);
    printf("Enter the year: ");
    scanf("%d", &y);

    // Error Handling performed with if-else statements
    if(m < 1 && m > 12) {
        printf("ERROR: Invalid month number");
    }
    else if(y < 0) {
        printf("ERROR: Invalid year");
    }
    output = countDays(m, y);       // Calling the function

    printf("\n%d days\n\n", output);    // Printing output array
}
```

```c
int countDays(int month, int year)  // Defining the function
{
    int days;
    // If month is not February, function uses regular day conditions
    if(month != 2) {
        if(month == 1 || month == 3 || month == 5 || month == 7 ||
           month == 8 || month == 10 || month == 12) {
            days = 31;              // months with 31 days
        }
        else if(month == 4 || month == 6 || month == 9 || month == 11) {
            days = 30;              // months with 30 days
        }
    }
    else if(month == 2) {           // if February, check for leap year
        if(year % 4 == 0 && year % 400 == 0) {
            days = 29;
        }
        else {
            days = 28;
        }
    }
    return days;                    // Return days variable to main
}
```

Figure 6: Days in a month calculator

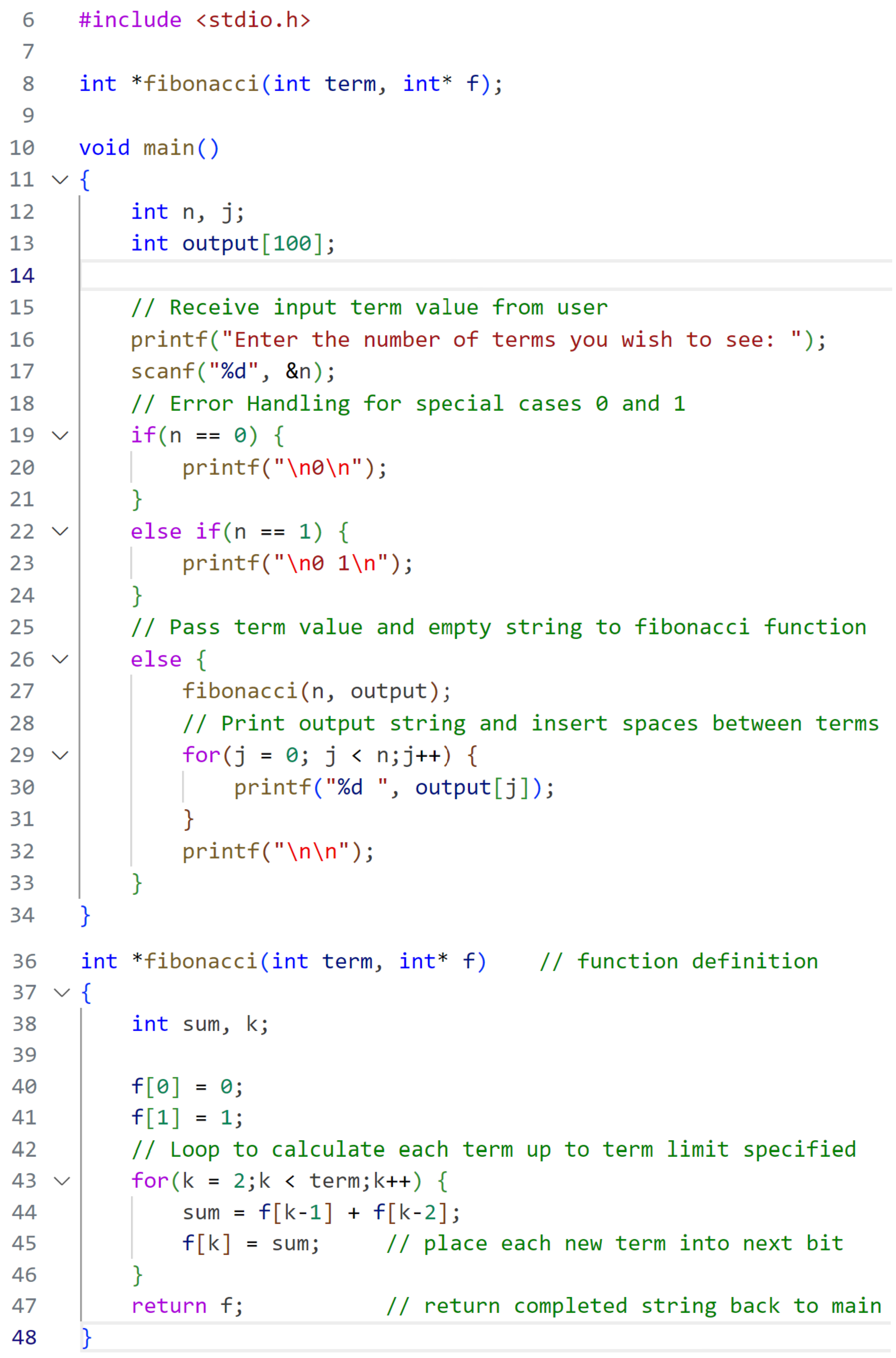
Program 3 functions by taking the input year and month from the user and cycles through a series of if else statements to (1) check if the month is not february, if it is, it sets the number of days to the constantly true values of that month and (2) if it is february, it sets the number of days by determining if the month was in particular leap year. This is important because the only special case for this program to do any thorough calculations is for February. In every other case, the days of a particular month are constant no matter the input year.



```
r374@raspberrypi:~/lab6 $ gcc Lab6_3.c -o Lab6_3.exe
r374@raspberrypi:~/lab6 $ ./Lab6_3.exe
Enter the month number: 2
Enter the year: 2000

29 days

r374@raspberrypi:~/lab6 $ ./Lab6_3.exe
Enter the month number: 2
Enter the year: 1900

28 days

r374@raspberrypi:~/lab6 $
```

Figure 7: Test Case Outputs for Program 3

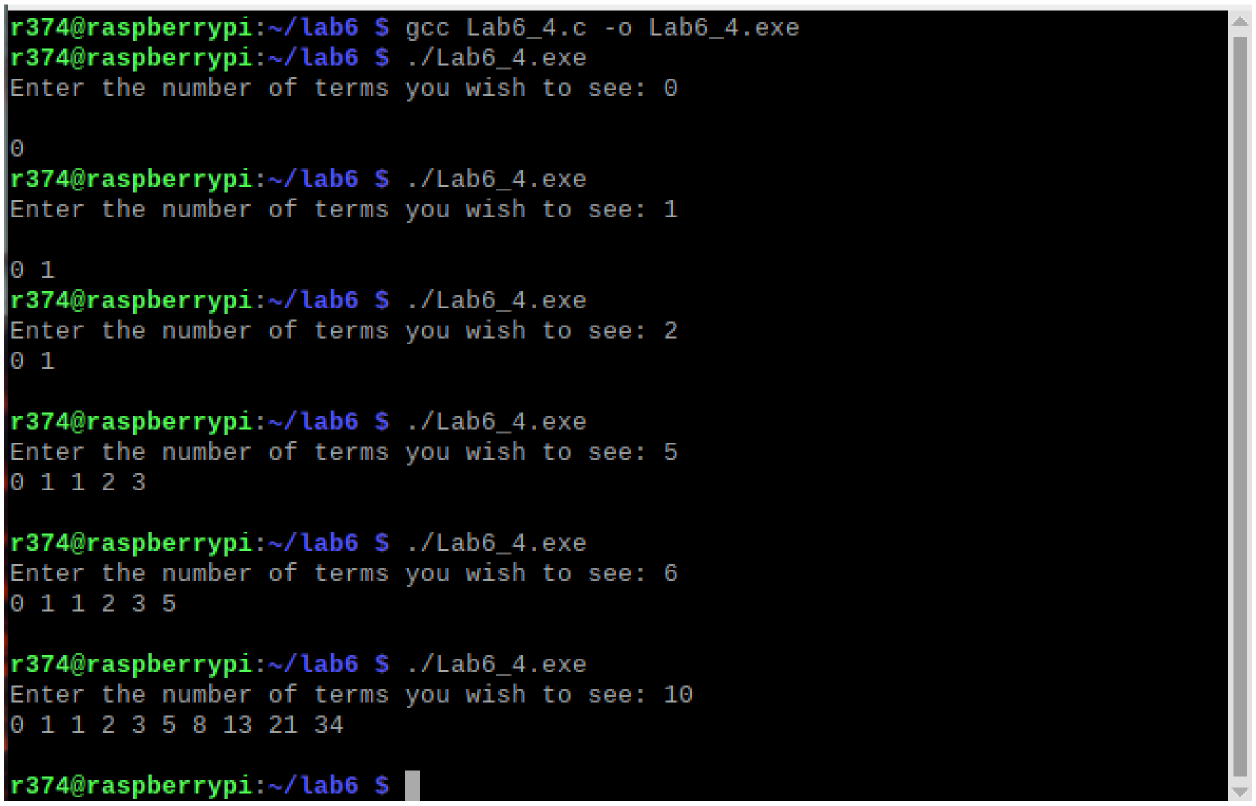
Program 4



```c
#include <stdio.h>

int *fibonacci(int term, int* f);

void main()
{
    int n, j;
    int output[100];

    // Receive input term value from user
    printf("Enter the number of terms you wish to see: ");
    scanf("%d", &n);
    // Error Handling for special cases 0 and 1
    if(n == 0) {
        printf("\n0\n");
    }
    else if(n == 1) {
        printf("\n0 1\n");
    }
    // Pass term value and empty string to fibonacci function
    else {
        fibonacci(n, output);
        // Print output string and insert spaces between terms
        for(j = 0; j < n;j++) {
            printf("%d ", output[j]);
        }
        printf("\n\n");
    }
}
int *fibonacci(int term, int* f)    // function definition
{
    int sum, k;

    f[0] = 0;
    f[1] = 1;
    // Loop to calculate each term up to term limit specified
    for(k = 2;k < term;k++) {
        sum = f[k-1] + f[k-2];
        f[k] = sum;     // place each new term into next bit
    }
    return f;           // return completed string back to main
}
```

Figure 8: Fibonacci Program Code

In the Fibonacci Program, the user is prompted for the number of terms they would like to receive. Then that value and an empty string is passed to the pointer function ‘fibonacci’ and the entire array of the series up to that term is calculated using a recursive method and placed in that empty string it receives. The recursive method involves the first 2 terms of the array being set to 0 and 1 respectively. Using a for loop, the sum of the next term is calculated using these first 2 terms and setting that sum to the next bit in the array. By using the variable k as the placeholder for the address of the particular empty string f (see f[k-1]) and incrementing it after every iteration, the string is produced in a recursive manner, needing to be constantly altered and called over and over again until it is created to its desired form. The outputs of this program are below.



```
r374@raspberrypi:~/lab6 $ gcc Lab6_4.c -o Lab6_4.exe
r374@raspberrypi:~/lab6 $ ./Lab6_4.exe
Enter the number of terms you wish to see: 0

0
r374@raspberrypi:~/lab6 $ ./Lab6_4.exe
Enter the number of terms you wish to see: 1

0 1
r374@raspberrypi:~/lab6 $ ./Lab6_4.exe
Enter the number of terms you wish to see: 2
0 1

r374@raspberrypi:~/lab6 $ ./Lab6_4.exe
Enter the number of terms you wish to see: 5
0 1 1 2 3

r374@raspberrypi:~/lab6 $ ./Lab6_4.exe
Enter the number of terms you wish to see: 6
0 1 1 2 3 5

r374@raspberrypi:~/lab6 $ ./Lab6_4.exe
Enter the number of terms you wish to see: 10
0 1 1 2 3 5 8 13 21 34

r374@raspberrypi:~/lab6 $
```

Figure 9: Test Case Outputs for Fibonacci Program

Conclusion

After completing this lab, I was able to strengthen my knowledge of function calling, construction, and recursive methods in C programming. In addition, I learned about the subtle differences between sending a data type and data type pointer to a function and a function pointer in practice. This type of repetitive practice with multiple small problems have really helped cement the concepts I learned in class.

In the future, I would like to apply my growing grasp of these concepts to slightly more complex problems.