Amy Yu
UID: 905138432
CEE/MAE M20
October 15, 2019

Homework 2

1. Take a date as input and return the day of the week

1.1 Introduction

The goal in this problem is to take the user's input of a year, month, and day, and prints out the day of the week. The day of the week will be calculated using the algorithm given in the homework problem.

1.2 Model and Methods

The script first prompts the user to enter the year and checks if it is valid. It also checks if it is a leap year, as that affects the validity of the days in February. Then, the user is prompted to enter a month using three letters and the program checks if it is valid. These letters are converted to uppercase using the `upper` function. A `switch` statement is used to give each month a numerical value, and printing an `error` statement if the month is not valid. Next, the user is prompted to enter a day, and an `error` is outputted if it is invalid. The algorithm given in the problem is used to determine the day of the week of the date that the user entered. The `mod` function is used in this algorithm to calculate the remainder when two numbers are divided. This was used in the equation in order to separate the last two digits of the year. The `floor` function is also used to round the number down. This was used in the equation to separate the first two digits of the year. Another `switch` statement is used to give each numerical value a day of the week, and this is printed using the `fprintf` function:

```
fprintf('%s %02d %04d %s %s', b, day, a, 'is a', dayName, '.')
```

The formatSpec `%s` means that it is a string and `%02d` means that it is an integer and prints two digits with a leading zero before it. `%04d` has a similar meaning but with four digits printed.
Lastly, the program checks if it is a Saturday or Sunday, and prints "`Yay, it is a weekend!`" if true.

1.3 Calculations and Results

When the program is executed and the user enters 2019 as the year, oct as the month, and 09 as the day, the following is outputted to the screen:

```
Please enter the year as YYYY (e.g. 2008):
2019
```

```
Please enter the month as MMM (e.g. FEB):
oct
Please enter the day as DD (e.g. 06):
09
OCT 09 2019 is a Wednesday.
```

The program successfully identifies October 19, 2019 as a Wednesday and prints the date in the right format, with the month in uppercase letters and the day with a leading zero.

When a weekend date is chosen, such as September 5, 1992, the following is outputted:

```
Please enter the year as YYYY (e.g. 2008):
1992
Please enter the month as MMM (e.g. FEB):
sep
Please enter the day as DD (e.g. 06):
05
SEP 05 1992 is a Saturday.
Yay, it is a weekend!
```

The program successfully identifies the date as a Saturday and prints the "`Yay, it is a weekend!`" message.

1.4 Discussion

This problem demonstrated the use of the `error` function. Many factors had to be considered when determining if a date is valid, such as a leap year and the number of days in a month. If-else statements were used to ensure that the program prints out the correct result. Switch statements were used to assign numerical values or names to the months and days of the week. By testing many different dates, it can be seen that the formula used to determine the day of the week is accurate.

2. Neighbor Identification

2.1 Introduction

The goal in this problem is to determine all the neighbors of a particular cell in a linearly-indexed grid with M rows and N columns. The number of neighbors depends on whether or not the target cell is on the edge or in the middle of the grid, and this will be shown in the output.

2.2 Models and Methods

The script first prompts the user to enter the number of columns and rows in the grid. This is checked for validity by using an `if` statement to see if M is less than 3, N is less than 3, or if they are not integers. The integer part is done by checking if the remainder of M or N divided by one is zero. Next, the user is prompted to enter P, the number of the cell to find neighbors for. P is checked if it is an integer and if it is within the bounds of the array. The program then uses if-else statements to identify if P is one of the corner cells or if it on one of the edges. A logical is set to false in the beginning of the program, and then changed to true if P is along a wall. First, the left wall is checked by seeing if P is less than or equal to M. If P is one of the corners, then there are three neighbors. Otherwise there are five neighbors. Next, the top wall is checked by seeing if the remainder of P divided by M is one, and the corners are excluded. The top wall will also have five neighbors. The same thing is done with the bottom wall by checking if the remainder of P divided by M is zero, excluding the corners, which will also have five neighbors. Lastly, the right wall is checked by seeing if P is greater than M*N-M. The corners will have three neighbors and the others will have five. All the other cells in the middle will have eight neighbors. An example of how the neighbors are printed is shown below:

```
fprintf('%d %d %d\n', 2, (1+M),(2+M))
```

The program ends by checking if P is a corner cell and printing '`Corner node`' if it is true.

2.3 Calculations and Results

When the program is run with M as 4, N as 6, and P as 1, the following grid should be modeled:

1	5	9	13	17	21
2	6	10	14	18	22
3	7	11	15	19	23
4	8	12	16	20	24

The following is printed to the screen:

```
Enter number of rows in the array:
4
Enter number of columns:
6
```

```
Enter cell to find neighbors:
1
2 5 6
Corner node
```

This shows that the program correctly identifies P as a corner node and on the left wall, and it also prints out the correct neighbors.

When the program is run again with the same M and N values, but with P as 15, the following is printed:

```
Enter number of rows in the array:
4
Enter number of columns:
6
Enter cell to find neighbors:
15
14 16 11 19 10 12 18 20
```

This shows that the method for determining if the cell is not a wall works, as the correct neighbors are printed.

Lastly, the program is tested using a different sized grid, where M is 5, N is 9, and P is 30.

1	6	11	16	21	26	31	36	41
2	7	12	17	22	27	32	37	42
3	8	13	18	23	28	33	38	43
4	9	14	19	24	29	34	39	44
5	10	15	20	25	30	35	40	45

The following is printed to the screen:

```
Enter number of columns:
9
Enter cell to find neighbors:
30
25 24 29 35 34
```

The program prints out the correct neighbors and identifies the cell as on the bottom wall.

2.4 Discussion

This program prints out all the neighbors of a given cell in a linearly-indexed grid. Many factors had to be taken into consideration when writing this code, such as if the cell is on the edge of the grid or if the cell is a corner. It was difficult figuring out how to identify the cells as on the wall. The mod function had to be used to determine if the cell was on the top or bottom wall. It was also slightly difficult trying to print the values, as there would be a different number of neighbors depending on the position of the cell. Everything had to be kept in terms of P, N, and M to ensure that the program would work for grids of all sizes.

3. Quadratic Function

3.1 Introduction

The goal in this problem is to print out the minimum and maximum values of a quadratic function on a certain interval. The user enters the coefficients and the upper and lower bounds of the function.

3.2 Models and Methods

The script first prompts the user to enter the values of L, R, a, b, and c. It checks if L is greater than R, and prints an error message if true. It also checks if a is 0, and if that is true, it prompts the user to enter the value of a again. Then, the values of L and R when plugged into the quadratic equation are stored as variables, as they will be used later. Since it is a quadratic equation, it will be a parabola. If the value of a is negative, the parabola points downward, and if the value of a is positive, the parabola points upward. The equation -b/2a is used to find the x-value of the vertex, and this is stored as the minimum x value if the parabola points down. This is plugged into the equation and stored as the minimum value. However, the boundaries must be taken into account. The left and right boundaries are also plugged into the equation and checked if it is less than the original minimum value. If that is true, the minimum value is adjusted. The larger value for left and right boundaries is chosen for the maximum value. A similar method is done if the parabola points downward and the results are printed.

3.3 Calculations and Results

When the program is run using -5 for L, 5 for R, 1 for a, and 0 for b and c, the following is outputted:

```
Enter L:
-5
Enter R:
5
```

```
Enter a:
1
Enter b:
0
Enter c:
0
minimum value = 0.00
maximum value = 25.00
```

These values were chosen because they can easily be plotted by hand and checked. It is true that the vertex is the at (0, 0), and that is the minimum in the interval. The maximum would be at the intersection of the parabola and the boundary line, which is at 25.

The program was run again using more complicated values, and the following was printed:

```
Enter L:
6.2
Enter R:
10.3
Enter a:
-2.4
Enter b:
6.2
Enter c:
7
minimum value = -183.76
maximum value = -46.82
```

In this example, neither the minimum or the maximum values are the vertex. This shows that the program correctly checks the possible cases. These values were checked on a graphing calculator, and shows that the program accurately computes the minimum and maximum values.

3.4 Discussion

The program ensures that the equation entered is a quadratic and forms a parabola by preventing the user from entering 0 as the value of a. This makes it easier to determine the minimum and maximum values, as the curve cannot change direction multiple times. The results are printed with two decimal points. Even though this is not necessary when the user inputs whole numbers and nice values, this gives us enough significant figures to check for accuracy if the user inputs decimal values.