Duncan Di Mauro
UID: 805163177
CEE/MAE M20
October 15, 2019

Homework 2

1 Input Date and Return the Day of the Week

1.1 Introduction

The goal in this problem is to create a code that takes a date as an input and returns the day of the week corresponding to that date. The day of the week will be calculated using Gauss's Algorithm for any date on the Gregorian calendar ranging from year 0001 to year 9999 (assuming the Gregorian calendar can extend backwards past October 1582). Following this calculation, the day of the week will be printed to the screen.

1.2 Models and Methods

The script first obtains a string assigned to the variable `year` by using a call to the `input` function. It then assigns the value of the input to the variable `year_value` via the `str2double` function. Conversions like this one are necessary, as opposed to obtaining an integer via the input, in order to create a custom error message when the user does not input a number. Using the `if` and `elseif` functions, the script then performs multiple checks to ensure that the inputs are within the proper bounds. These checks include: using `length(year)` to ensure that the length of the string is 4 characters long, using the `mod` function to check that the input can be converted to an integer number (i.e. has no remainder when divided by 1), and finally checking that the value of the integer is between 1 and 9999. If any of these conditions are not met, the script returns an error stating "Invalid Input."

Next, the script obtains a string assigned to the variable `month` using a call to the `input` function. The `upper` function is used to change the string to all uppercase letters. Then, the `switch` function is used to assign numerical values for each month and their maximum number of days. Additionally, the `otherwise` function is used to produce an error message if the input is not a properly abbreviated month. A sample is shown below:

```
switch month
    case 'JAN'
        month_value = 1;
        final_day = 31;
    case 'FEB'
    …
otherwise
        error('Invalid Input');
end
```

For February, an `if` statement is used to assign `final_day` to either 28 or 29, depending on if it is a leap year. The rules used for leap years include: It is a leap year if the year is divisible by 4 and either (1) not divisible by 100, or (2) divisible by 400.

The script then obtains a string assigned to the variable `day` using a call to the `input` function, and obtains another variable `day_value` using the `str2double` function. Just like the checks used for the `year` input, an `if` statement is used to confirm that the input for `day` has a length of 2, and that the value is an integer not less than 1 and not greater than the value of `final_day`.

Next, the day of the week is calculated using Gauss's Algorithm, as shown in MATLAB formatting below:

```
w = mod((d + floor(2.6*m - 0.2) + y + floor(y/4) + floor(c/4) - 2.*c), 7)
```
(1)

where

- `w` is day of the week (0 is Sunday, 6 is Saturday),
- `d` is day of the month,
- `m` is the shifted month calculated as ((month + 9) mod 12) + 1,
- `y` is the last 2 digits of the year
- `c` is the first 2 digits of the year

and the value for `y` is decreased by 1 if the month is January or February (this was accounted for by subtracting 1 from `year_value` if `month_value` equals 1 or 2). A value for `d` was obtained via day_value, `m` was obtained using the equation ((month_value + 9) mod 12) + 1, `c` was obtained by using `floor(year_value/100)`, and `y` was obtained using `year_value - (c*100)`.

Finally, the month, day, year, and day of the week were printed to the command window in the following format: `JAN 01, 2000 is a Saturday` with an extra message "Yay, it is a weekend!" if the inputted date is on a weekend. An example is shown below:

```
if w == 1
    fprintf('\n%s %s %s %s', month, day, year, 'is a Monday.');
end
```

1.3 Calculations and Results

When the program is executed, and the user inputs "2020" for year, "feb" for month, and "29" for day, the following output is printed to the screen:

```
Please enter the year as YYYY (e.g. 2008): 2020
Please enter the month as MMM (e.g. FEB): feb
Please enter the day as DD (e.g. 06): 29

FEB 29 2020 is a Saturday.
Yay, it is a weekend!.
```

1.4 Discussion

In this problem the main things I learned were how to use and apply the `mod`, `floor`, `upper`, `if`, `switch`, `||` ,and `str2double` functions. This problem was very straightforward to solve given Gauss's equation, along with the rules that determine whether a year is a leap year. The hardest part was understanding how the `switch` function worked, but other than that, there are no interesting results to discuss.

2 Neighbor Identification

2.1 Introduction

The goal in this problem is to find the values for all neighbor cells of a certain cell `P` in a rectangular array. Values for the number of rows `M`, the number of columns `N`, and the desired cell `P` are inputted by the user, and the values of the neighbor cells are printed to the screen.

2.2 Models and Methods

The script first obtains 2 strings assigned to the variables `Rows` and `Columns` by using two calls to the `input` function (the inputs in this script are set as strings in order to allow for custom error messages to appear when a non-numerical value is entered, just as in the first homework problem). Next, `str2double` is used to assign the values of `Rows` and `Columns` to the variables `M` and `N`, respectively. An `if` statement is then used to check that `M` and `N` are both integers and are both greater than or equal to 3. If they do not meet these conditions, an error message is printed.

The script then uses a call to the `input` function to prompt the user to enter a value for the desired cell, which is assigned to the variable `P_string`, followed by a line of code assigning the numerical value of `P_string` to the variable `P`. An `if` statement is then used to check that `P` is an integer, is greater than or equal to 1, and is less than or equal to `M*N`. If this is not true, an error message is printed.

The script then calculates the values for all neighbors of `P` (assuming that `P` has all 8 neighbors) and assigns them to variables "`Neighbor1`" through "`Neighbor8`." This is done using the equations shown below, which were deduced using logic and a simple understanding of rectangular array layout:

```
Neighbor1 = P - M;      %Left of P
Neighbor2 = P + M;      %Right of P
Neighbor3 = P + 1;      %Below P
Neighbor4 = P - 1;      %Above P
Neighbor5 = P - M - 1;  %Upper-Left of P
Neighbor6 = P - M + 1;  %Lower-Left of P
Neighbor7 = P + M - 1;  %Upper-Right of P
Neighbor8 = P + M + 1;  %Lower-Right of P
```

It is fine that we obtain values for all 8 neighbors, even if they might not exist, since our goal is to simply print the neighbors that do exist. `P` will have different neighbors depending on if it is located along the top, left, bottom, or left wall, if it is located on the top-left, top-right, bottom-left, or bottom-right corner, or if it is not touching any wall. Therefore, we can utilize the `if` and `elseif` functions to print the neighbors of `P` depending on which category `P` fits into. The determination of P's category was found using the equations below (these equations were derived using logic and a basic understanding of rectangular array layout):

- if P = 1 P is Upper-Left corner
- if P = M*(N-1) + 1 P is Upper-Right corner
- if P = M P is Lower-Left corner

- if P = M*N P is Lower-Right corner
- if P <= M P is against Left Wall
- if P > M*(N-1) P is against Right Wall
- if mod(P - 1, M) = 0 P is against Upper Wall
- if mod(P, M) = 0 P is against Lower Wall

A sample of the way the `if` function was used to print the relevant neighbor values is shown in pseudocode below:

```
if P == 1 (If P is the Upper-Left Corner)
    fprintf('\n%s%i %i %i', 'Corner node: ', Neighbor3, Neighbor2, Neighbor8)
elseif P is in some other category
    fprintf(Printing the values for the Category's Neighbors)
end
```

2.3 Calculations and Results

When the program is executed, and the user enters 20 for M, 13 for N, and 241 for P, the following output is printed to the screen:

```
Enter an integer value for the number of rows, M: 20
Enter an integer value for the number of columns, N: 13
Please enter the value of the desired cell, P: 241

Corner node: 221 222 242
```

2.4 Discussion

This problem was also very straightforward, however the most difficult part was creating the equations in the `if` statements for the different possible categories of `P` (different corners/walls). I learned how to approach problems like these, in that I should separate the value of a variable (in this case, `P`) into categories based on the different results (neighbors) that the value produces. I wonder if there is a simpler way to write this script that does not involve creating variables for every single possible neighbor and creating different `print` statements depending on the location of `P`. Otherwise, there are no unique results to discuss or compare.

3 Quadratic Function

3.1 Introduction

The goal of this problem was to write a script that solicits real numbers `L`, `R`, `a`, `b` and `c` as inputs from and prints the minimum value and maximum value of

$$f(x) = ax^2 + bx + c \tag{2}$$

on the interval [`L`,`R`]. Additionally, if `a` = 0, the script should print, "`a should not be zero for a quadratic function`" and should prompt the user to enter another nonzero value of *a*.

Furthermore, if `L > R`, the program should pop out an error `Please make sure L is less than R`.

3.2 Models and Methods

Like the other two problems, all inputs for this problem were assigned to be strings, and then converted into numerical values via `str2double`. The script starts out by prompting the user to enter values for `L` and `R`, which are originally assigned to `L_string` and `R_string`. Then, the script uses the `if` function to produce an error if the numerical value of `L_string` is greater than `R_string`.

Next, a `while` loop is used to prompt the user to enter a value for `a` until `a` is not equal to 0. This is done by assigning an arbitrary variable to be false (in our case, that variable is `FLAG`) , and while that variable is false, the user must input a value for `a`. Once the value of `a` is nonzero, the statement becomes true and the loop ends. This is shown below:

```
FLAG = false;
while ~FLAG
    a_string = input('Enter a value for a: ', 's');
    if str2double(a_string) == 0
    fprintf('\na should not be zero for a quadratic function\n\n')
    elseif str2double(a_string) ~= 0
       FLAG = true;
    end
end
```

The script then uses 2 calls to the `input` function to obtain strings representing values for `b` and `c`. Next, str2double is applied to `L_string`, `R_string`, `a_string`, `b_string`, and `c_string`, and the values are assigned to variables `L`, `R`, `a`, `b`, and `c` respectively. The string then uses the `if` function along with the `isnan` function to produce an error if any of these values are not a number. This is shown below:

```
if isnan(L) || isnan(R) || isnan(a) || isnan(b)|| isnan(c)
    error('Please only enter real numbers')
end
```

The variable `x` is then assigned to be all values starting at `L` and going until `R` with increments of 0.001. This code for this is as follows: `x = L:0.001:R`
The variable `f` was then set equal to equation (2), and 2 more variables `minimum` and `maximum` were set equal to the minimum and maximum values of `f`, which were obtained using the `min` and `max` functions.

Finally, the minimum and maximum values of the function were printed with 3 significant figures on the right side of the decimal, since values for `x` were separated by increments of 0.001.

3.3 Calculations and Results

When the program is executed, and the user enters 12.5 for L, 29.3 for R, 6 for a, 1.234 for b, and 8.8 for c, the following output is printed to the screen:

```
Enter a value for the lower bound, L: 12.5
Enter a value for the upper bound, R: 29.3
Enter a value for a: 6
Enter a value for b: 1.234
Enter a value for c: 8.8

Minimum Value = 961.725
Maximum Value = 5195.896
```

3.4 Discussion

In this problem, the main things I learned were how to construct a 1-dimensional array, how to use a `while` loop, and how to use the `isnan` function. The 1-dimensional array was used for `x`, and allowed `x` to go from `L` to `R`. I chose for the `x` values to be separated by increments of 0.001 because it is impossible for `x` to be separated by infinitely small increments, and I assumed 0.001 to be small enough when dealing with values that have an absolute value larger than 1. Otherwise, there are no interesting results to compare, and this problem is very straightforward.