Linh Tang

CEE/MAE M20

UID: 205542275

Oct 16, 2020

HOMEWORK 2

1. Assignment submission:

1.1 Part 1

1.1.1 Introduction

This part is calculating and returning the submission date after taking the date a student's homework is assigned, and how long it takes to finish the homework as inputs from users. Besides, the program is designed to check the validity of all the inputs before the calculation.

1.1.2 Model and methods

The script first asks the user enters inputs for day month and year for the assignment date. Those inputs for day, month, year must be numbers only; this means there are no letters and special characters. For this requirement, inputs are entered as string data type, then calling string functions str2double(), isnan() and contains() to make sure only numbers are obtained. Str2double and isnan are used to see if the string can be converted to a number; contains() is used to exclude some special characters since str2double can take a comma, decimal point, leading + - as a part of number.
The following codes is checking validity for day input; month and year will be similar.

```
get_d = input('Enter day: ', 's');
day = str2double(get_d); %converting string to number
if (isnan(day))
    error ('Invalid input. Only numbers for day.');
elseif (contains(get_d,'.')||contains(get_d,',')||
contains(get_d,'+'))
    error ('Invalid input. Only numbers for day.');
end
```

Besides, there are boundaries for day, month, and year. The problem assumes there are 30 days in each month, 12 months in a year within 360 days. Day, month, year, and duration cannot be negative.

```
if (day <0 || day > 30)
    error ('Invalid day input. Day shoulbe be 1-30\n');
end
if (month <1 || month > 12)
    error ('Invalid month input. Month shoulbe be 1-12\n');
end
if (year < 1)
    error ('Invalid year input. Year must be a positive
number.\n');
end
```

To start with the calculation, the amount of time needed to get the homework done first is converted to the format year, month, day. The value is divided in three cases. The first case is when the duration less than 30 days; this means it is not enough to form a month, therefore it is counted toward “days” only, zero for month and year. The second case is when the duration greater than 30 days but less than 360 days; therefore, there might be enough numbers of days to form months, but not for a year. The last one has a value big enough to form year when it is greater and equal to 360 days. For the calculation, floor() and mod() functions are called; the calculation basically fills up year, month and then day.

```
if (time < 30)
    d = time;
    m = 0;
    y = 0;
elseif (time >= 30 && time < 360)
    d = mod(time,30);
    m = floor(time/30);
    y = 0;
elseif (time >= 360)
    y = floor(time/360);
    remainder = mod(time, 360);
    m = floor(remainder/30);
    d = mod(remainder,30);
end
```

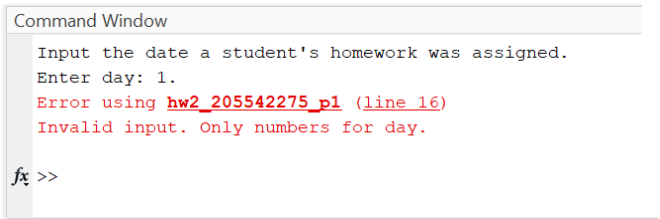
After converting the duration from days to year, month, day, those values are added to the assignment date to get the date the homework is complete. One more step is performed to correct the submission date if the result is invalid. When the day greater than 30, it can form another month. When the month is greater than 12, it will add one to the year.

```
%Getting submission dates
sub_day = day + d;
sub_month = month + m;
sub_year = year + y;

%Checking new dates if it is valid, Correct it if not
if(sub_day > 30)
    sub_day = sub_day - 30;
    sub_month = sub_month + 1;
end
if (sub_month > 12)
    sub_month = sub_month - 12;
    sub_year = sub_year + 1;

end
```

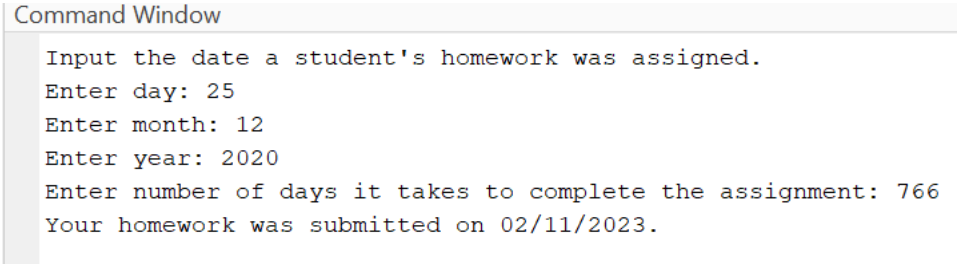
For displaying the submission date, fprintf is used, and %0.2d to print out leading zero if day is one digit.

```
fprintf ('Your homework was submitted on
%02d/%02d/%04d.\n\n', sub_month, sub_day,sub_year);
```

1.1.3 Results



The assignment date is 12/25/2020; it took the student 766 days to finish the homework. The program returns a correct date 02/11/2023 as the submission date.



1.1.4 Discussion

The program for this part run well and gave out correct dates. In the future, the script can be more advanced if timing (hours, minutes, seconds) is added since it does not take so much time (years or months) to finish the homework. One more thing can be improved is that months can be displayed as string (JAN for 1) when displaying the date.

1.2 Part 2

1.2.1 Introduction

This purpose of this part is getting the due date of the homework from the user. This input will be compared with the submission date to determine if the homework is submitted on time, late, or early. Based on the determination, the program returns how many days the student is late and what score will be given.

1.2.2 Methods and model

The script gets the input for due date and check its validity in a same way as part 1. One more thing needs to be considered is the due date cannot be earlier than the date the homework is assigned. A simple way to compare the dates is everything converted to number of days. The one is earlier has less total of days and vice versa. To check validity, the due date is compared with the assignment date.

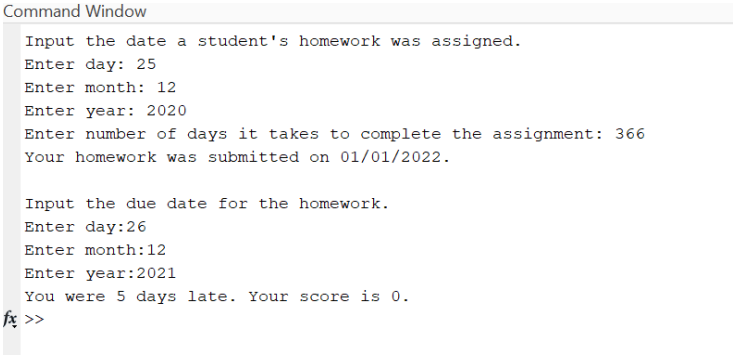
```
% Converting dates to total number of days
total_ass = day + month*30 + year*360;
total_due = due_d + due_m*30 + due_y*360;
total_sub = sub_day + sub_month*30 + sub_year*360;

% Due date cannot be earlier than assignment date
if (total_due <= total_ass)
    error('Invalid due date input. Due date cannot be earlier
than the assignment date.\n');
end
```

To check if the student is late of not, the submission date is compared with the due date. Late days is only calculated when the total days of due date less than the total days of submission date.

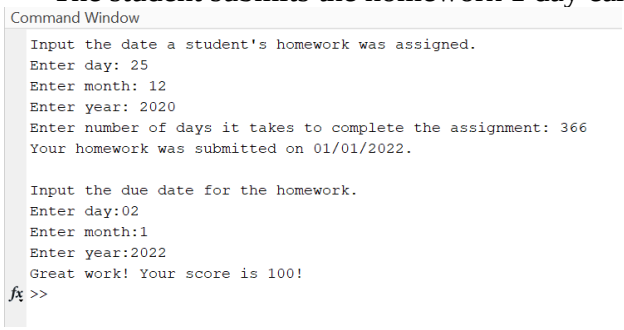
```
if(total_sub > total_due)
    late = total_sub - total_due;
    if(late >= 5)
        fprintf('You were %d days late. Your score is
0.\n',late);
    else
        score = 100 - (late*10); % reduced 10% for each day
late
        fprintf('You were %d days late. Your score is %d.\n',
late, score);
    end
else
    fprintf('Great work! Your score is 100!\n');
end
```

1.2.3 Results

When the due date is 12/26/2021, but the student submits it on 01/01/2022. Student is late for 5 days and then get 0.



The student submits the homework 1 day early.



1.2.4 Discussion

Similar to the discussion in part 1, some features might be added to get a more advanced version.

2. Neighbor identification

2.1 Introduction

The goal of this script is finding neighbors of a specific cell in a matrix. The target cell that is needed to find neighbors, and the size of the matrix are obtained from users.

2.2 Model and methods

Before starting the calculation, the size of matrix and the cell are obtained from users by using input function. The matrix is valid is when its row and column are not less than 3. Based on the structure of the matrix, the matrix starts with cell 1 at top left corner and ends at cell row*col at bottom right corner.

```
M = input('Enter number of rows (M): ');
if(M <= 3)
    error('Invalid input. M must be greater than 3.\n');
end

N = input('Enter number of columns (N): ');
if(N <= 3)
    error('Invalid input. N must be greater than 3.\n');
end

P = input('Enter the cell (P): ');
if(P < 1 || (P > M*N))
    error('Invalid input. P must be from 1 to M*N.\n');
end
```

There are three positions where the cell locates returns different number of neighbors. Four corner cells return, each one returns three neighbors. Cells at first row, last row, first column, last column (excluding 4 corner cells) return 5 neighbors. The rest of cells return 8 neighbors.

For the corner cells, those four cells need to be defined based on M(row) and N (column).

```
last_num = M*N;
first = 1;
second = M;
third = last_num - M + 1;
fourth = last_num;
```

With other two cases, floor() and mod() functions are used to determine the relation of the position of the cell to the size of the matrix.

```
floor_M = floor(P/M);
mod_M = mod(P,M);
```

A cell is located at the middle when its row is from row 2 to row M -1; this means mod(P, M)greater than 1. Its column is from col 2 to col N-1; this means floor(P/M) is from 1 to N-2.

```
if((floor_M > 0)&& (floor_M < N-1)&&(mod_M > 1))
…
end
```

For the second case, the cells at the edges except four corners are identified in a similar way. Cells at the first row divide by M always give 1 as the remainder while quotient is from 1(col 2) to N-2(col N-1).

```
if((mod_M == 1)&& (floor_M > 0) && (floor_M < N -1))
    …
end
```

Cells at the last row is divisible by M while quotient is from 1(col 2) to N-2(col N-1).

```
if((mod_M == 0)&& (floor_M > 0) && (floor_M < N -1))
    …
end
```

Cells at the first column divide by M always give 0 as quotient while the remainder is greater than 1; it moves from cell at second row till reaching last row (cell at the last row is divisible by M, mod is 0).

```
if((floor_M == 0) && (mod_M >1))
…
end
```

Cells at the last column divide by M always give N-1 as quotient while the remainder is greater than 1.

```
if((floor_M == N-1) && (mod_M >1))
…
end
```

Neighbor cells are calculated based on given values such as P, M by using arithmetic operations (+, -).

Neighbor cell can be either P ± M, P ± 1, or P ± M ± 1.

For example:

```
% last row
if((mod_M == 0)&& (floor_M > 1) && (floor_M < N))
    n1 = P - M;
    n2 = P + M;
    n3 = P - 1;
    n4 = n3 - M;
    n5 = n3 + M;
    fprintf ('%d %d %d %d %d', n1, n2, n3, n4, n5);
end
```

At the end, the number of neighbors is printed out depends on each case by calling fprintf function; “Corner node” is for corner nodes case.

2.3 Results

- Cell 4

```
Command Window
  Enter number of rows (M): 4
  Enter number of columns (N): 6
  Enter the cell (P): 4
  CornerNode.
  Cell 4 has neighbors: 3 8 7.
fx >>
```

- Cell 13

```
Command Window
  Enter number of rows (M): 4
  Enter number of columns (N): 6
  Enter the cell (P): 13
  Cell 13 has neighbors: 9 17 14 10 18.
fx >>
```

- Cell 19

```
Command Window
  Enter number of rows (M): 4
  Enter number of columns (N): 6
  Enter the cell (P): 19
  Cell 19 has neighbors: 15 23 18 14 22 20 16 24.
fx >>
```

2.4 Discussion

The scrip gives out exact neighbor cells for cell P for different cases. This problem can be solved using array instead; the value of neighbor will be returned as the value at that index using arr(row, col).

3. Type of number

3.1 Introduction

The goal of this problem is to determine an input, a 6-digit number, if it is symmetrical, increasing, decreasing, or neither. The input must contain 6 digits and numbers only.

3.2 Model and methods

First, the script asks the user to enter a 6-digit number by using input function. Since all integers are divisible by 1, mod(num,1) is used here to make sure the input only contains digit. To check whether the number has 6 digits or not, the number is converted to a string, then using the length() function.

```
num = input('Enter 6 digit number:');

if (mod(num,1) ~= 0)
    error('Invalid input. Enter numbers only.');
end
```

```
s = num2str(num);
if (length(s) ~= 6)
    error('Invalid input. Enter a number with 6 digits.');
end
```

Each digit is separated one by one backwards, from the last one to the first one of a number. When dividing the number by 10, the remainder gives us the last digit by using the mod() function while the floor() function gives us the rest of digits. Therefore, mod() and floor() functions are used repeatedly to get each digit. After the separation, each digit is stored into a variable.

```
sixth = mod(num,10);
f1 = floor(num/10);

fifth = mod(f1,10);
f2 = floor (f1/10);

fourth = mod(f2,10);
f3 = floor (f2/10);

third = mod(f3,10);
f4 = floor (f3/10);

second = mod(f4,10);
f5 = floor(f4/10);

first = mod(f5,10);
f6 = floor(f5/10);
```

After the separation, the comparison is performed to determine which type of the number is. The number is symmetric when the first digit is the same with the sixth digit; the second digit is the same with the fifth digit; the third digit is the same with the fourth digit.

The number is increasing if the first is less than the second; the second is less than the third; the third is less the fourth, so on until we reach the last digit. For the decreasing number, the similar logic is used, but using “greater than”.

```
if(first == sixth && second == fifth && third == fourth)
    % Symmetrical
    fprintf('%d is symmetrical.\n', num);

elseif (first < second && second < third && third <fourth
&& fourth <fifth && fifth < sixth)
    % Increasing
    fprintf('%d is increasing.\n', num);

elseif (first > second && second > third && third > fourth
&& fourth > fifth && fifth > sixth)
    % Decreasing
    fprintf('%d is decreasing.\n', num);
else
    % Neither
    fprintf('%d is not either increasing or decreasing or
symmetrical.\n', num);
end
```

3.3 Result

- Test for validity

```
Command Window
  Enter 6 digit number:12345
  Error using hw2_205542275_p3 (line 11)
  Invalid input. Enter a number with 6 digits.

fx >>
```

```
Command Window
  Enter 6 digit number:123e45
  Error using hw2_205542275_p3 (line 11)
  Invalid input. Enter a number with 6 digits.

fx >>
```

- Increasing number

```
Command Window
  Enter 6 digit number:235689
  235689 is increasing.
fx >>
```

- Symmetrical number

```
Command Window
  Enter 6 digit number:124421
  124421 is symmetrical.
fx >>
```

- Decreasing number

```
Command Window
  Enter 6 digit number:754321
  754321 is decreasing.
fx >>
```

- Neither

```
Command Window
  Enter 6 digit number:122334
  122334 is not either increasing or decreasing or symmetrical.
fx >>
```

3.4 Discussion

The most difficult thing in this problem was figuring out how to separate six digits one by one. Using mod () and floor () functions work fine in this problem, for 6 digits. However, if the number contains greater than 6 digits, such as 10 digits, a loop is needed to repeat steps.