Jared Rivera

804603106

CEE/MAE M20

January 24, 2017

# Homework 2

## 1 Day-of-the-Week Calculator

### 1.1 Introduction

The goal of this problem is to write a program that takes dates as input, and returns the day of the week corresponding to that date as an output. The dates are to be taken as strings in MMM/DD/YYYY format and inputs are checked to make sure they are valid in the sense that they meet the format and exist. The results will be printed to the screen in the form MMM DD YYYY is a (insert day of the week).

### 1.2 Model and Methods

The first step is to get an input date from the user. Directly after, numerous validity checks must be done to ensure the date entered exists. The input is taken as a string, and the string is converted into a number using the string to number function like so

```
YN=str2num(Y)
```

To make sure the entry is also an integer in the case of day and year, the following is used

```
length(YN)~=1
```

Parameters around the year range are set up using logical statements such as

```
if YN<0 || YN>9999 || mod(YN,1)~=0
```

Parameters around format of each entry (day, month, year) are set using string length functions. To ensure input of months only includes actual months, parameters are set up for each month individually, as shown below

```
strcmp(M, 'JAN')
```

In a similar fashion, months with the same amount of days are grouped together and the parameter for days is set. The example of February, including an account for leap years, is shown

```
if strcmp(M,'FEB')==1 && leapyear==1 && DN>29
```

If the user input ever doesn't meet the conditions outlined, the built in MATLAB error function is called on, as is shown here when the user doesn't input within the correct date range

```
error('Please enter a year between 1AD and 9999AD');
```

Next, the algorithm for day of the week assignment is set up. First, the variables within the algorithm are defined in terms of the variables already present in the script. For example, the c and y variables that break up the year into its first two and last two integers are defined as:

```
y=mod(YN,100);
c=floor(YN/100);
```

The algorithm is then written and its value is a number between 0 and 6. To then assign this value to a day, a set of `ifelse` statements is written with print options inside that assign the appropriate day.

### 1.3 Calculations and Results

When the program is run, the user will initially be prompted for input

```
Please enter the month as MMM (e.g. JAN): JAN

Please enter the day as DD (e.g. 07): 24

Please enter the year as YYYY (e.g. 2016): 2017
```

Once the inputs in this example are entered, the following is printed

```
JAN 24 2017 is a Tuesday
```

### 1.4 Discussion

The intricacy of this problem is in its validation. Prompting for user input, writing the algorithm, and printing the results to the command window is all comparable to work done in week 1. In this problem, since the user enters more complex data and many more constraints lie on the problem, it is harder to enforce proper input. This problem is an excellent tool in learning the mathematical and functional relationships of many operators (i.e. mod and floor) as well as logical expressions. There are dozens of "if"s in this problem to account for the multitudes of possible input. Understanding how to express conditions in terms of conditionals and in a clever form using operators is key to ensuring no input is unaccounted for, though in this case there may be some errors that could slip through.

## 2 Binary Addition

### 2.1 Introduction

The goal of this problem is to write a program that can add integers. The addition will be performed in binary rather than decimal and will only be done using logical gates rather than the built in addition function. The program will take two integer inputs between 0 and 7, convert them to binary, perform logical addition, then output the binary form of each integer, the binary sum, and the decimal sum.

### 2.2 Model and Methods

The first step is to prompt the user for input of two integers between or equal to 0 and 7. To ensure the input lies in that range, a logical expression is used as follows

```
if A_in<0 || A_in>7 || mod(A_in,1)~=0
```

```
error('Please follow the instructions more carefully!');
end
```

In this case, the logical expression checks to see if the input doesn't meet the criteria. If so, it outputs an error message.

Next, the input values are converted into three binary components, $A_0$, $A_1$, and $A_2$. These values are found using mathematical expressions and a knowledge of the definition of each one. $A_2$ Is the number of times 4 goes into the number, $A_1$ is the number of times 2 goes into the number, and $A_0$ is the number of times 1 goes into the number. They are calculated as follows

```
A2=floor(A_in/4);
A1=floor(mod(A_in/2,2));
A0=mod(A_in,2);
```

These expressions make sense in that `A2` Takes the integer amount 4 divides into the number, `A1` takes that remainder form that action and checks if its divisible by 2, and `A0` checks for any leftover in the 1s place. Next, addition is carried out. Here sum bits and carry-over bits are defined as they were in class

```
S0=xor(xor(A0,B0),C0);
C1=and(A0,B0) || and(A0,C0) || and(B0,C0);
```

The addition is carried out left to right as it is when done by hand with the final sum bit being equal to the final carry over bit. The final decimal value is then found by multiplying each sum bit by its corresponding power of two

```
DEC=8*S3+4*S2+2*S1+1*S0
```

Finally, the results are printed to the command window in the format specified by the question.

### 2.3 Calculations and Results

When the program is run, the user enters two inter values A and B:

```
Input an integer value between 0 and 7: 5

Input an integer value between 0 and 7: 7
```

The value is then computed and the following is printed to the command window:

```
The decimals provided are A=5 and B=7

The conversion of A to binary: 101

The conversion of B to binary: 111

A plus B in binary is: 1100

A plus B in decimal is: 12
```

## 2.4 Discussion

This problem demonstrates the power of logical expressions and the ability to assign a bit to a characteristic of a system. The logical gates are enough to perform a fairly complicated task and this can be extrapolated out toward higher ranges of addition or more complex operations like multiplication. It also gives the programmer a greater knowledge of how the machine interacts with scripts, as those 0 and 1 bits are all it operates off of. The reason this solution only required up to the third power of two in its binary accounting is that any combination of two numbers between 0 and 7 can be represented by a combination of two to the powers of 0, 1, 2, and 3. One great addition (no pun intended) to this script would have been a FOR loop. It could have calculated all the sum and carry over bits without the programmer having to write out the same command for every bit in the converted integers.