Morteza Kazemi | 97243054
Amir Masoud Shaker | 97243081
12/12/2021

Homework #7

Q1

Dual-tone multi-frequency signaling (DTMF) is a signaling system used in telecommunication which uses voice-frequency band over telephone lines between communication devices and switching centers. it uses **eight audio frequencies** in pairs of two to represent 16 signals (0 to 9, A to D, # and *). These signals do not need operators on long-distance circuits because they are audible tones in voice-frequency range and can be transmitted through electrical repeaters and amplifiers, and over radio and microwave links.

Multi-frequency signaling (MF) is using a mix of two pure tone (pure sine wave) sounds for signaling. One of the MF signaling protocols is for in-band signaling between switching centers, where long-distance telephone operators use it to route the next destination telephone number. Based on the success with using MF to establish long-distance telephone calls, dual-tone multi-frequency signaling was developed for <u>end-user signaling without the need of operators</u>.

In order to prevent interfering consumer telephones and routing between telephone switching centers, DTMF frequencies differ from MF signaling protocols between switching centers. Also as DTMF signaling is often transmitted in-band with audio signals, it has limits for timing, frequency deviations, etc.

The DTMF telephone keypad is shown in the right. Rows and columns show the frequency components. As an illustrative example, pressing 1, produces a mix of a 697 Hz and a 1209 Hz tone that is decoded by the switching center to determine the keys pressed by the user.
Symbols #,*,A, B, C and D were added to the keypad to access computers and automated response systems. However, in the end, A to D keys were dropped from most phones, they could be used to assert certain privilege and priority for calls, control network, select menu items, etc.

DTMF keypad layout.

The telephone keypad system replaced rotary dial(dial pulse) with the help of DTMF. It was also used in caller ID systems to transfer caller information, in cassettes for information encoding and in television broadcasting and controlling remote transmitters.

The code for the question is in the following page:

```
clc;
clear;

command = input('enter the input string: ', 's'); % get user input as a string
command_len = size(command, 2); % the length of the string

% process the string char by char
for i=1:command_len
    process(command(i), i);
end

% _____________________________________________________
% _____________________ functions _____________________
% _____________________________________________________
function process(char, char_index)
    % processes one character of the string

    [row_freq, col_freq] = lookup(char);
    if(char_index ~= 1)
        pause(0.900); % the first char does not need a pause before being played
    end

    create_sound(row_freq, col_freq);
    pause(0.500); % pause execution for 500 milliseconds
    clear sound;

end

function create_sound(row_freq, col_freq)
    % creates the sine wave of the given frequencies and plays it.

    samp_freq = 16000; % sampling frequency in Hz
    samp_dur= 5; % sampling duration in Sec
    t = 0 : 1/samp_freq : samp_dur; % time samples

    w1=sin(2*pi*t*row_freq); % first sine wave with the row frequency
    w2=sin(2*pi*t*col_freq); % second sine wave with the column frequency
    soundsc(w1+w2,samp_freq) % playing the dual tone signal
end

function [row_freq, col_freq] = lookup(char)
    % a lookup table that returns row and column (high and low) frequencies
    % needed to create the proper sound for each key.

    switch char
        case {'1','2', '3', 'A'}
            row_freq = 697;
        case {'4','5', '6', 'B'}
            row_freq = 770;
        case {'7','8', '9', 'C'}
            row_freq = 852;
        case {'*','0', '#', 'D'}
            row_freq = 941;
        otherwise
            error(['invalid character in the input!' char])
    end

    switch char
        case {'1','4', '7', '*'}
            col_freq = 1209;
        case {'2','5', '8', '0'}
            col_freq = 1336;
        case {'3','6', '9', '#'}
            col_freq = 1477;
        case {'A','B', 'C', 'D'}
            col_freq = 1633;
        otherwise
            error(['invalid character in the input!' char])
    end
end
```

Q2

In the Piano_simulation.m file, we simulated two octaves of piano (starting at middle C).
It has both white and black keys (which means all the notes in each octave) and they are placed similar to real piano.
First we create an audiorecorder object which helps us to record the sound of piano and we record for 20 seconds. We set a name for the audio file that we're going to make and using audiowrite, we write the audio file in the directory that we are in it.
Then we start our most important part of code.
We create an empty figure which responds to the keyboard.
The write the function key, that uses a switch case structure in it to determine the right frequency for each key.
We start at C4 and end at C6# which contains a little bit more than two octaves.
We have a formula for computing the frequency for each node and that is:
freq = 2 ^ ((n-49)/12) * 440
In the frequency_values.m file, we compute frequencies for all the notes between C4 and C6# just to make sure that the values in our key function our correct.
Finally, we create the wave for each note and pass it to sound function which converts the signal data to sound.
Three audio files named audio1, audio2, audio3 are created.

Resources:
Piano - File Exchange - MATLAB Central (mathworks.com)
Piano key frequencies - Wikipedia
Record and Play Audio - MATLAB & Simulink - MathWorks Deutschland

Piano_simulation:

```
clc;
clear;

Fs = 1E+4;
nBits = 24;
nChannels = 1;

my_audio = audiorecorder(Fs, nBits, nChannels);

disp('Start recording')
recordblocking(my_audio, 20);
disp('End Recording');

filename = 'audio3.wav';
audiowrite(filename, getaudiodata(my_audio), Fs);

figure('keypress', @key);
% make and empty figure window which responds to the keyboard


function key(~, event)
    switch event.Key
        case 'q', freq = 261.6256;   % C4 (middle C)
        case '2', freq = 277.1826;   % C4#
        case 'w', freq = 293.6648;   % D4
        case '3', freq = 311.1270;   % D4#
        case 'e', freq = 329.6276;   % E4
        case 'r', freq = 349.2282;   % F4
        case '5', freq = 369.9944;   % F4#
        case 't', freq = 391.9954;   % G4
        case '6', freq = 415.3047;   % G4#
        case 'y', freq = 440.0000;   % A4
        case '7', freq = 466.1638;   % A4#
        case 'u', freq = 493.8833;   % B4
        case 'z', freq = 523.2511;   % C5
        case 's', freq = 554.3653;   % C5#
        case 'x', freq = 587.3295;   % D5
        case 'd', freq = 622.2540;   % D5#
        case 'c', freq = 659.2551;   % E5
        case 'v', freq = 698.4565;   % F5
        case 'g', freq = 739.9888;   % F5#
        case 'b', freq = 783.9909;   % G5#
        case 'h', freq = 830.6094;   % G5#
        case 'n', freq = 880.0000;   % A5
        case 'j', freq = 932.3275;   % A5#
        case 'm', freq = 987.7666;   % B5
        case 'k', freq = 1046.502;   % C6
        case 'l', freq = 1108.731;   % C6#
    end

    % a function that sets the value of frequency for each note
    % using a switch case

    wave = sin(freq*2*pi*[0:1/8192:.3]);
    % create the wave for each note using the formula

    sound(wave);
    % convert the signal data(wave) to sound

end
```

Frequency_values:

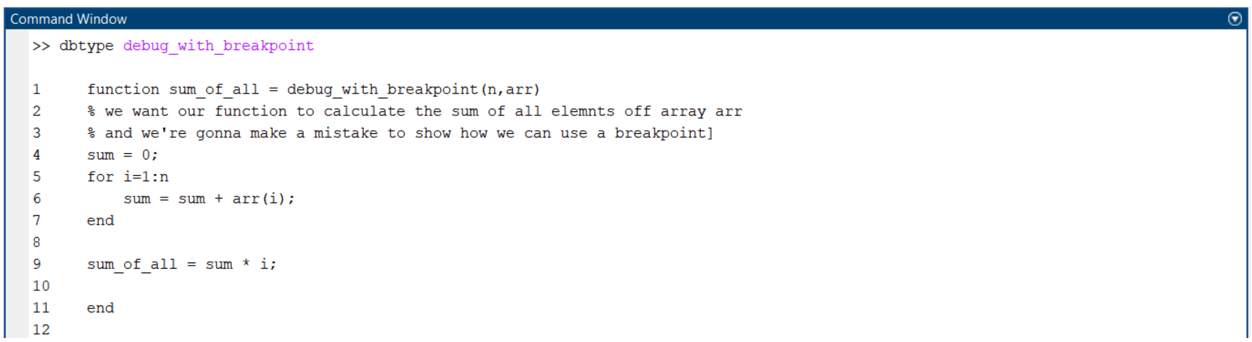
```
freqs = zeros(1,26);

for i=40:65
    freqs(i-39) = 2 ^ ((i-49)/12) * 440;
    % the formula for nth key frequency in piano
    % suppose (i-39) as n. it's just for indexing the freqs array better
end

for i=1:size(freqs,2)
    freqs(i)
    % show that our calculated frequencies are correct and exactly the same
    % as freq for each key in piano_simulation.m
end
```

Q3

How to debug an m-file with breakpoints:
First we should know that the debugging commands work on both functions and scripts.
Here is our function that we want to debug :

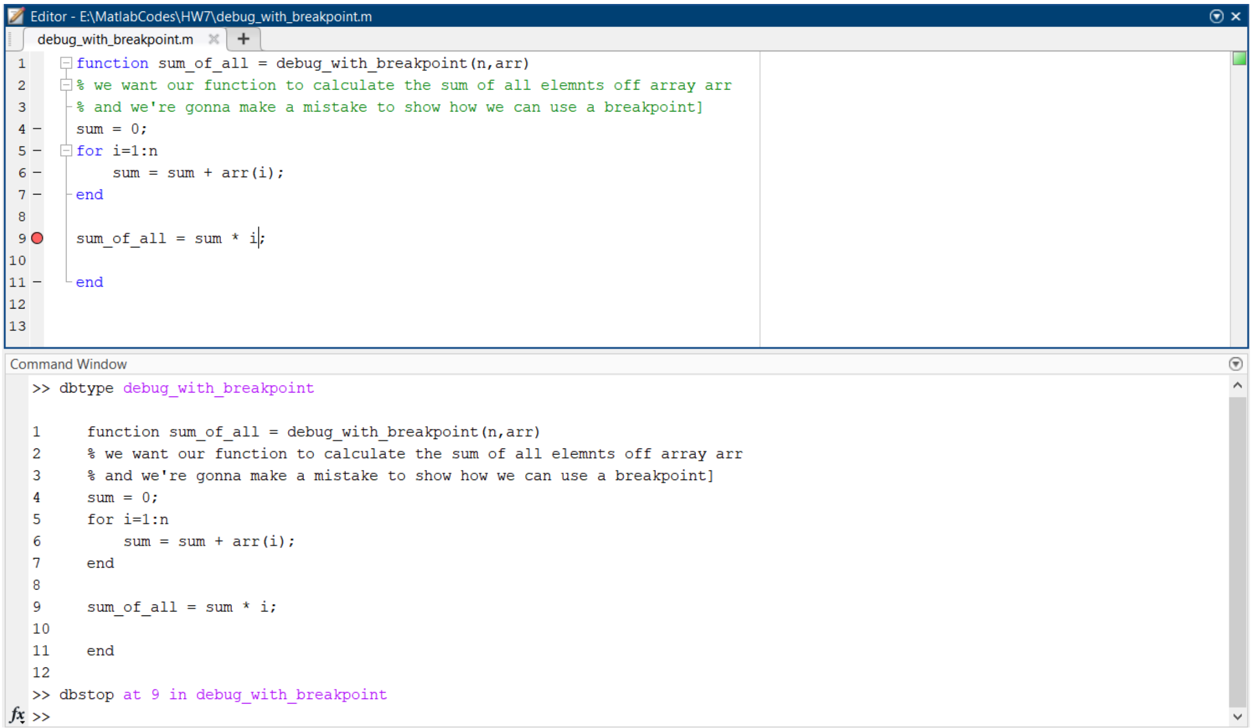
```
function sum_of_all = debug_with_breakpoint(n,arr)
% we want our function to calculate the sum of all elemnts off array arr
% and we're gonna make a mistake to show how we can use a breakpoint]
sum = 0;
for i=1:n
    sum = sum + arr(i);
end

sum_of_all = sum;

end
```

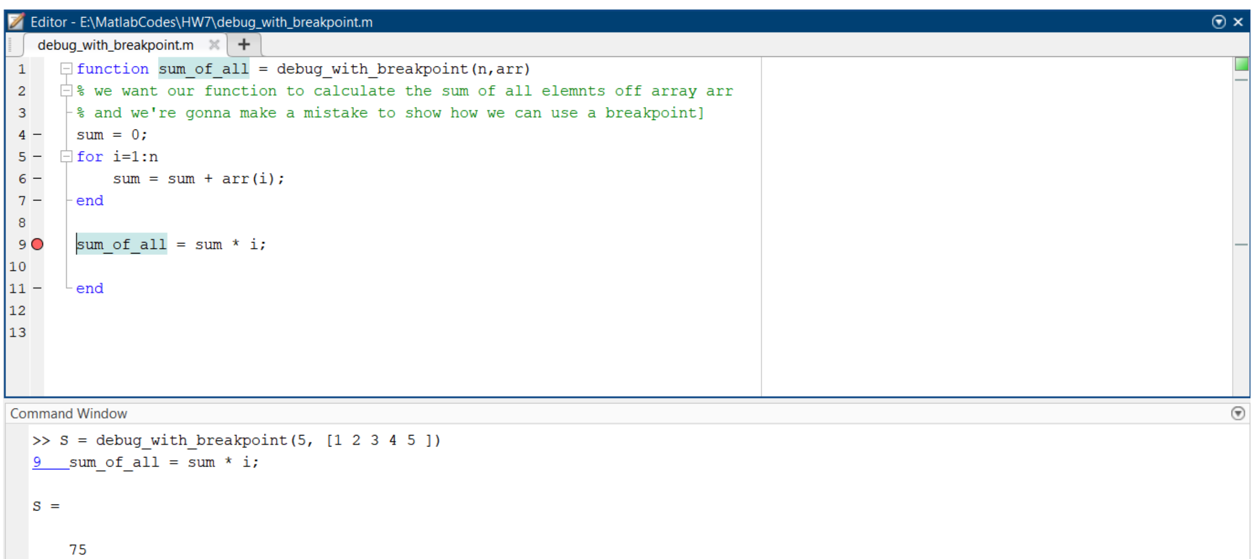
First we can use **dbtype** command to list the line numbers.



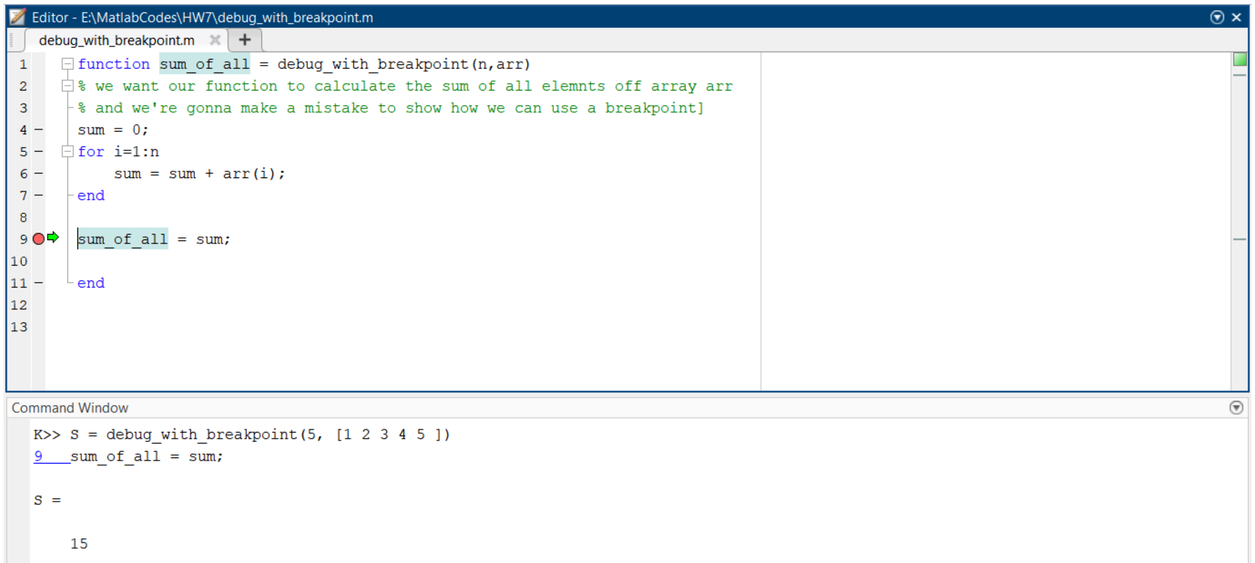
```
Command Window
>> dbtype debug_with_breakpoint

1     function sum_of_all = debug_with_breakpoint(n,arr)
2     % we want our function to calculate the sum of all elemnts off array arr
3     % and we're gonna make a mistake to show how we can use a breakpoint]
4     sum = 0;
5     for i=1:n
6         sum = sum + arr(i);
7     end
8
9     sum_of_all = sum * i;
10
11    end
12
```

Then we set our breakpoint in line that we want using **dbstop** command:



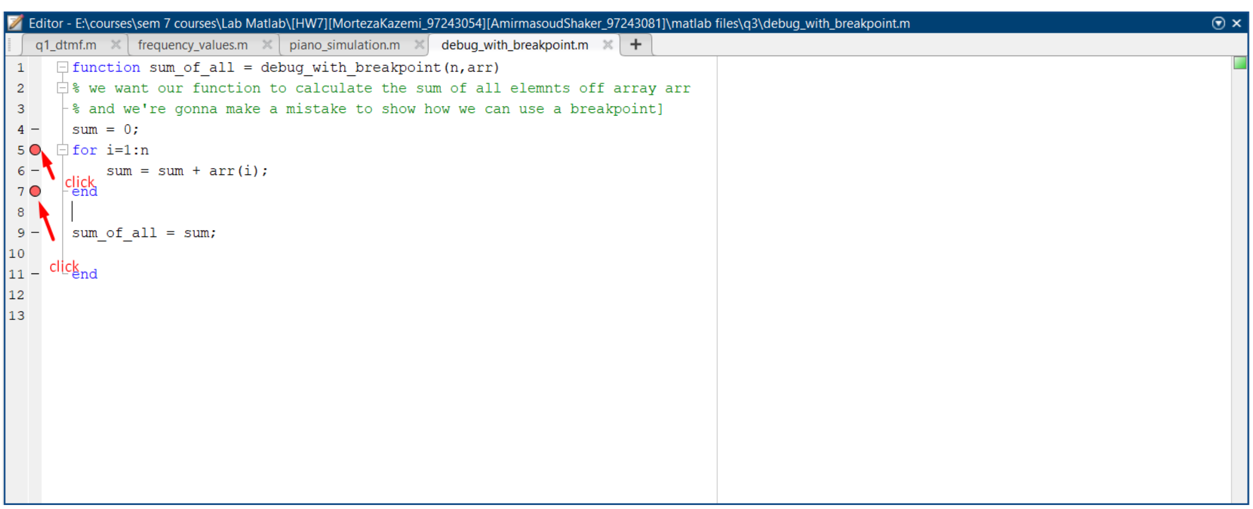


Now we can see that the multiply by i is additional and should be removed.
So we fix the error easily:

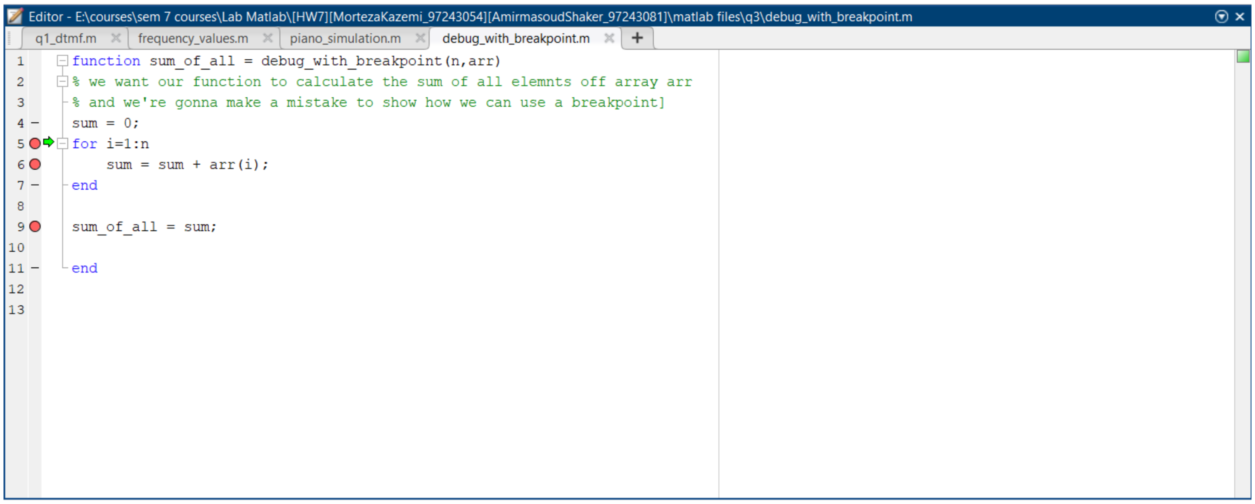


We can also use the Matlab GUI to debug. This is an easier way of debugging.

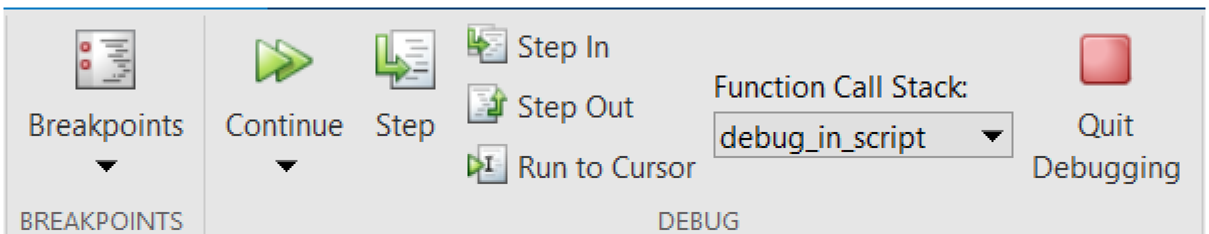
First, we can put the breakpoints by clicking on the left side of the editor pane. We can have multiple breakpoints in one file.



Then we run the program and see that it stops at a certain line of the code shown by a green arrow.



When in debug mode, we can see the following tab in the Matlab toolbar:



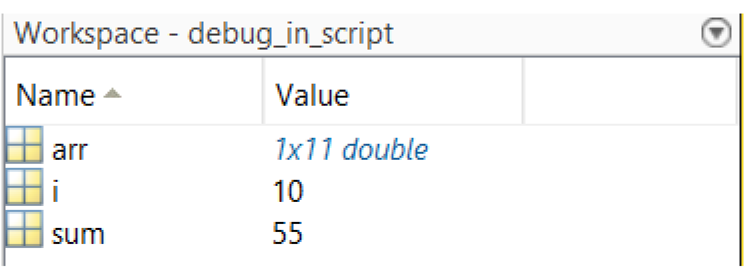
By clicking on Continue, we run the program to the next breakpoint.

By clicking on Step, the program executes one line of the code.

By clicking step in we can see through the functions we call and by clicking step out we can get out of the current scope in which we are.

Run to cursor could be an option to move faster. We need to click on a line of our code so that the cursor would be set there. Then, we can click on this option to execute the program upto that line of the code.

On the right side of the Matlab window, we can see the workspace pane:



We can see our variables values by double clicking on it or writing its name in the command window(or even writing more complex scripts to show a desired dimension of our variables).