ELE 338 – Preliminary Work 4

Anıl Karaca – 21728405

Q1)

Source Code:

```
org 100h
;ELE338 - Preliminary Work 4 - Question 1
;Anıl Karaca - 21728405
JMP start
msgShape: DB "Enter S/s for square or T/t for triangle:",0Dh,0Ah,24h
msgHeight: DB "Enter the height of the shape:",0Dh,0Ah,24h
msgError: DB "It is not a valid input",0Dh,0Ah,24h

;Prints "X" on the emulator screen
printX PROC
    MOV DX, 58h
    MOV AH, 2h
    INT 21h
RET
printX ENDP

;Prints space on the emulator screen
printSpace PROC
    MOV DX, 20h
    MOV AH, 2h
    INT 21h
RET
printSpace ENDP

;Prints new line on the emulator screen
printNewline PROC
    MOV DX, 0Dh
    MOV AH, 2h
    INT 21h
    MOV DX, 0Ah
    MOV AH, 2h
    INT 21h
RET
printNewline ENDP
```

```
;Print prompt msg and get the height
getHeight PROC
    CALL printNewline
    MOV DX, msgHeight
    MOV AH, 9h
    INT 21h
    MOV AH, 1h
    INT 21h
RET
getHeight ENDP

;Controls the operation for square shape
squareProc PROC
    ;Get the height of the shape properly
    promptHeight:
    CALL getHeight
    MOV AH, 0d ;Reset AH
    ;Check if the input is valid
    CMP AX, 3Ah ;39h is "9" so we check 3Ah
    JGE promptHeight
    CMP AX, 30h ;30h is "0"
    JLE promptHeight

    MOV BX, 0d ;Reset BX
    SUB AL, 30h ;Turn the ASCII to integer
    MOV BL, AL ;Store the height in BL
    ;Check if the input is 1
    CMP AL, 1d
    JZ oneSquare

    MOV CX, 0d ;Reset CX
    MOV CL, BL
    ;Print the first line
    CALL printNewline
    firstLine:
    CALL printX
    LOOP firstLine
```

```
    MOV CX, BX
    SUB CX, 2d
    CMP CX, 0d ;Check if the height is 2
    JZ ifTwo ;If it is 2 skip the middle part
    ;Print the middle lines
    middleLine:
    MOV SI, BX
    SUB SI, 2d

    CALL printNewline
    CALL printX ;Print the first X

    ;Print the spaces in between
    addSpace:
    CMP SI, 0d
    JZ doneSpace
    CALL printSpace
    DEC SI
    JMP addSpace

    doneSpace:
    CALL printX ;Print the second X
    LOOP middleLine

    ifTwo:
    MOV CX, 0d ;Reset CX
    MOV CL, BL

    ;Print the last line
    CALL printNewline
    lastLine:
    CALL printX
    LOOP lastLine
    CALL printNewline

    JMP terminateSquare

    oneSquare:
    CALL printNewline
    CALL printX
    ;JMP terminateSquare

terminateSquare:
RET
squareProc ENDP
```

```
;Controls the operation for triangle shape
triangleProc PROC
    ;Get the height of the shape properly
    promptHeightTri:
    CALL getHeight
    MOV AH, 0d ;Reset AH
    ;Check if the input is valid
    CMP AX, 3Ah ;39h is "9" so we check 3Ah
    JGE promptHeightTri
    CMP AX, 31h ;31h is "1"
    JLE promptHeightTri

    MOV BX, 0d
    SUB AL, 30h
    MOV BL, AL ;Store the height in BL

    MOV CX, 0d ;Reset CX
    MOV CL, BL
    DEC CL
    ;Print the first line
    CALL printNewline
    firstLineSpace:
    CALL printSpace
    LOOP firstLineSpace
    CALL printX

    CMP BL, 2d ;Check if the height is 2
    JZ ifTwoTri ;If it is 2 skip the middle part

    MOV SI, 1d
    MOV BP, BX
    CALL printNewline
    ;Print the middle lines of a triangle
    middleLineTri:
    MOV CL, BL
    SUB CX, SI
    DEC CX
    ;Print the space before the first X
    firstSpace:
    CALL printSpace
    LOOP firstSpace
```

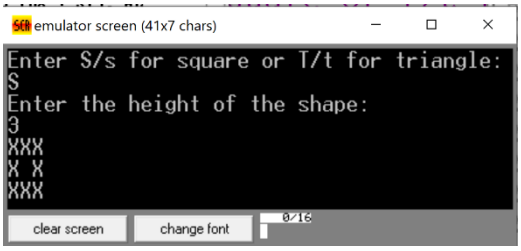
```
    ;Print the first X
    CALL printX

    MOV DI, SI
    ADD DI, DI
    DEC DI
    MOV CX, DI
    ;Print the space between the first and the second X
    secondSpace:
    CALL printSpace
    LOOP secondSpace

    ;Print the second X and a new line
    CALL printX
    CALL printNewline

    INC SI
    DEC BP
    CMP BP, 2d
    JNZ middleLineTri
    JMP ifNotTwo

    ifTwoTri:
    CALL printNewline
    ifNotTwo:
    ;Print X on the last line
    MOV CL, BL
    ADD CL, CL
    DEC CL
    lastLineX:
    CALL printX
    LOOP lastLineX

RET
triangleProc ENDP
```

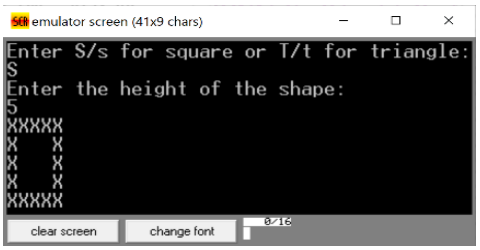
```
start:
;Print prompt msg to get the shape
MOV DX, msgShape
MOV AH, 9h
INT 21h
;Get the shape input
MOV AH, 1h
INT 21h

;Check if the input is "S" or "s"
CMP AL, 53h
JZ square
CMP AL, 73h
JZ square
;Check if the input is "T" or "t"
CMP AL, 54h
JZ triangle
CMP AL, 74h
JZ triangle

;If the input is invalid print error msg
CALL printNewline
MOV DX, msgError
MOV AH, 9h
INT 21h
CALL printNewline
JMP start

square:
CALL squareProc
JMP terminate
triangle:
CALL triangleProc
JMP terminate



terminate:
ret
```

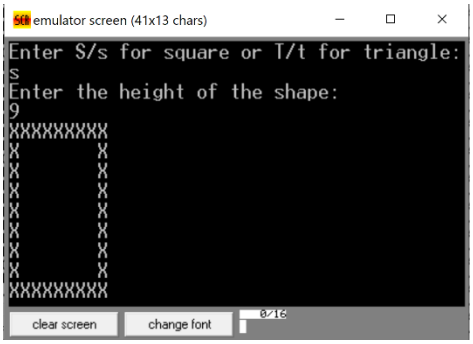
Screenshots:

Emulator Screen:

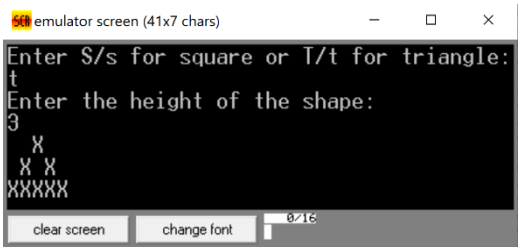
Square Examples:

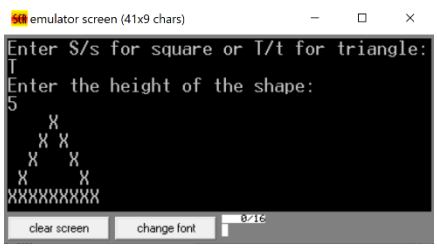


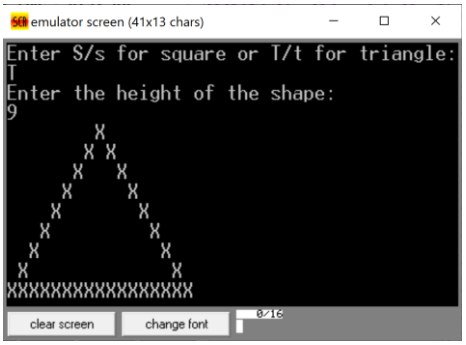




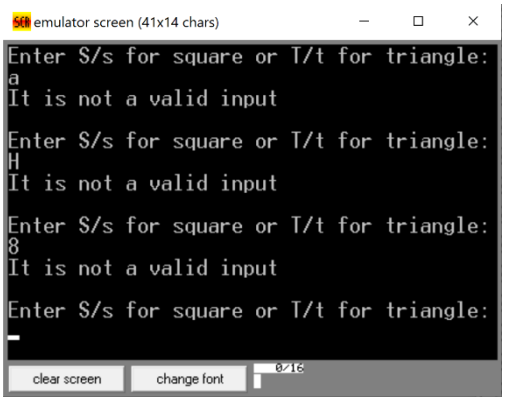
Triangle Examples:

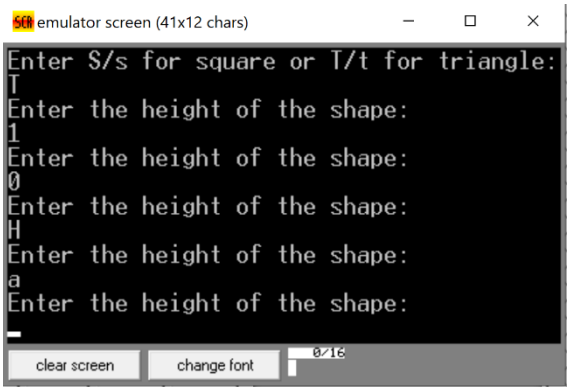


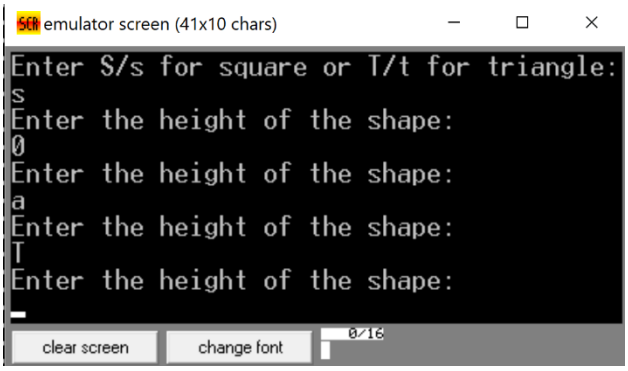




Invalid Case Examples:







Comments:

Firstly, program prompts user to enter a shape(a square or a triangle). If the entered input is not S/s or T/t, program prints "It is not a valid input" and keeps repeating the input prompt until user enters a valid input.

And then, program prompts user to enter a height value, this prompt keeps popping on the screen until a valid input is entered. The valid input for square and triangle are:

Square	1, 2, 3, 4, 5, 6, 7, 8, 9
Triangle	2, 3, 4, 5, 6, 7, 8, 9

"1" is not a valid answer for triangle because, triangle with a height of 1 doesn't make sense.

After a valid height value is entered, the algorithm starts printing the desired shape with the given height value. For explanation reasons let's say the height is denoted by H.

Squares:

For the first and the last lines in the squares, we print H number of "X" on the emulator screen. For the lines in the middle, firstly we print an "X", and then we print $H - 2$ number of "space" characters on the screen and finally we print an "X" one more time.

Since height of "1" and "2" are the exceptions of this algorithm, program handles them differently. If the height is "1" the program prints an "X" and then terminates the program. If the height is "2" the program skips the part where we print the lines in the middle.

For explanation reasons of the algorithm behind printing triangles, let's use one more variable denoted by n, this n value starts from 1 and each time we go down a line n increases by 1.

Triangles:

For the first line in the triangles, we print $H - 1$ number of "space" characters on the screen, and then print an "X". For the lines in the middle we first print $H - 1 - n$ number of "space" characters, and then we print the first "X", then we print $2n - 1$ spaces in between, finally we print the second "X". For the last line in the triangles, we print H number of "X" on the emulator screen.

Since height of "2" is an exception of this algorithm, program handles this situation differently. It basically skips the part where we print the lines in the middle.

Q2)

Source Code:

```
org 100h

;ELE338 - Preliminary Work 4 - Question 2

;Anıl Karaca - 21728405

JMP start

msgShape: DB "Enter S/s for square or T/t for triangle:",0Dh,0Ah,24h

msgHeightSquare: DB "Enter the height of the square (000-480):",0Dh,0Ah,24h

msgHeightTriangle: DB "Enter the height of the triangle (000-320):",0Dh,0Ah,24h

msgError: DB "It is not a valid input",0Dh,0Ah,24h


;Prints new line on the emulator screen

printNewline PROC

    MOV DX, 0Dh

    MOV AH, 2h

    INT 21h

    MOV DX, 0Ah

    MOV AH, 2h

    INT 21h

RET

printNewline ENDP


;This procedure takes 3 digit input and stores it in BP

formatInput PROC

    ;1st digit input

    MOV AH, 1h

    INT 21h

    SUB AX, 30h

    MOV BL, 100d

    MUL BL

    MOV BP, AX

    ;2nd digit input

    MOV AH, 1h

    INT 21h

    SUB AX, 30h

    MOV BL, 10d

    MUL BL

    ADD BP,AX

    ;3rd digit input

    MOV AH, 1h

    INT 21h


    MOV AH, 0d

    SUB AX, 30h

    ADD BP, AX

RET

formatInput ENDP


squareProc PROC

    ;Prompt height of the square

    promptHeightSquare:

    CALL printNewline

    MOV DX, msgHeightSquare

    MOV AH, 9h

    INT 21h

    CALL formatInput

    CMP BP, 480d ;Check if the input is valid

    JGE promptHeightSquare ;Keep looping until the input is valid


    ;Find starting point CX, DX

    MOV BX, 2d

    MOV DX, 0d

    MOV AX, BP

    DIV BX


    MOV CX, 320d ;Set CX to be half of screen width

    MOV DX, 240d ;Set DX to be half of screen height


    SUB CX, AX

    SUB DX, AX


    ;Find ending point SI, DI

    MOV SI, CX

    ADD SI, BP


    MOV DI, DX

    ADD DI, BP
```

```
;Open 640x480 screen
MOV AH, 0d
MOV AL, 12h
INT 10h
INT 10h

MOV AL, 3d ;Set the color of the pixels

;Print upper side
upperSide:
MOV AH, 0Ch
INT 10h
INC CX ;Update coordinate
CMP CX, SI
JLE upperSide

;Print right side
rightSide:
MOV AH, 0Ch
INT 10h
INC DX ;Update coordinate
CMP DX, DI
JLE rightSide

;Print bottom side
SUB SI, BP
bottomSide:
MOV AH, 0Ch
INT 10h
DEC CX ;Update coordinate
CMP CX, SI
JGE bottomSide
```

```
    ;Print left side
    SUB DI, BP
    leftSide:
    MOV AH, 0Ch
    INT 10h
    DEC DX ;Update coordinate
    CMP DX, DI
    JGE leftSide

RET
squareProc ENDP

triangleProc PROC
    ;Prompt height of the triangle
    promptHeightTriangle:
    CALL printNewline
    MOV DX, msgHeightTriangle
    MOV AH, 9h
    INT 21h
    CALL formatInput
    CMP BP, 320d ;Check if the input is valid
    JGE promptHeightTriangle ;Keep looping until the input is valid

    ;Find starting point CX, DX
    MOV BX, 2d
    MOV DX, 0d
    MOV AX, BP
    DIV BX
    PUSH AX ;Store this value so we can use it later

    MOV CX, 320d ;Set CX to be half of screen width
    MOV DX, 240d ;Set DX to be half of screen height

    SUB DX, AX
```

```
;Find ending point SI, DI
MOV SI, CX
ADD SI, AX

MOV DI, DX
ADD DI, BP

;Open 640x480 screen
MOV AH, 0d
MOV AL, 12h
INT 10h
INT 10h

MOV AL, 3d ;Set the color of the pixels

;Print right side
rightSideTriangle:
MOV AH, 0Ch
INT 10h
ADD DX, 2d ;Update coordinate
INC CX
CMP DX, DI
JLE rightSideTriangle

;Print bottom side
SUB SI, BP
bottomSideTriangle:
MOV AH, 0Ch
INT 10h
DEC CX ;Update coordinate
CMP CX, SI
JGE bottomSideTriangle
```

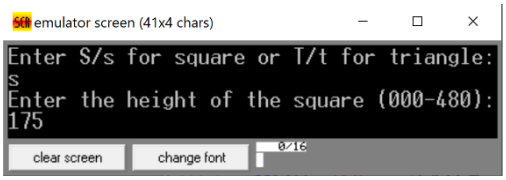
```
    ;Print left side
    POP AX
    ADD SI, AX
    SUB DI, BP
    MOV AL, 3d ;Set the color of the pixels again
    leftSideTriangle:
    MOV AH, 0Ch
    INT 10h
    SUB DX, 2d ;Update coordinate
    INC CX
    CMP DX, DI
    JGE leftSideTriangle

RET
triangleProc ENDP

start:
;Print prompt msg to get the shape
MOV DX, msgShape
MOV AH, 9h
INT 21h
;Get the shape input
MOV AH, 1h
INT 21h
```

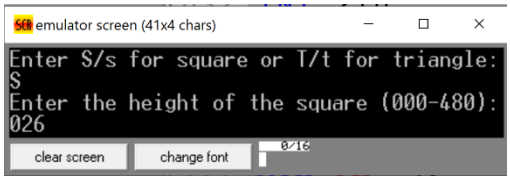
```
;Check if the input is "S" or "s"
CMP AL, 53h
JZ square
CMP AL, 73h
JZ square
;Check if the input is "T" or "t"
CMP AL, 54h
JZ triangle
CMP AL, 74h
JZ triangle

;If the input is invalid print error msg
CALL printNewline
MOV DX, msgError
MOV AH, 9h
INT 21h
CALL printNewline
JMP start

square:
CALL squareProc ;Print square by calling it's procedure
JMP terminate
triangle:
CALL triangleProc ;Print triangle by calling it's procedure
JMP terminate

terminate:
ret
```

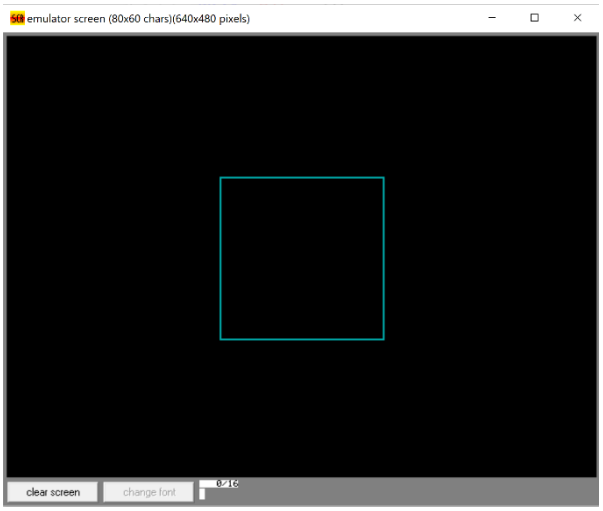
Screenshots:

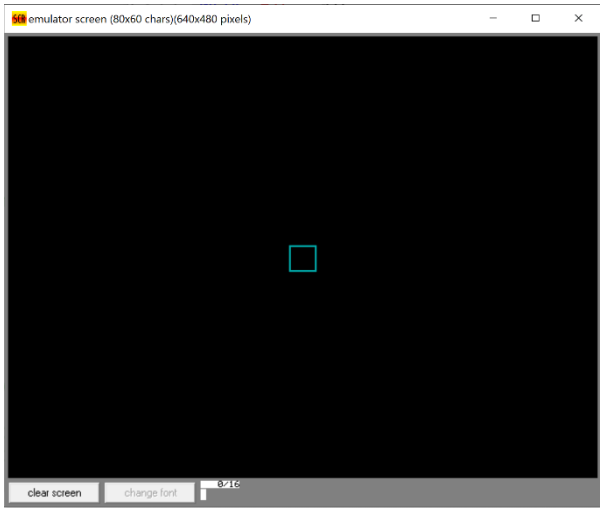
Emulator Screen:

Square Examples:

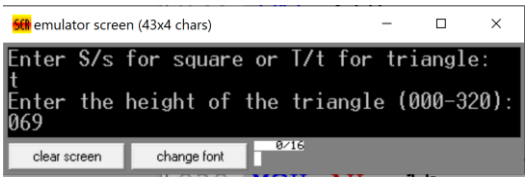


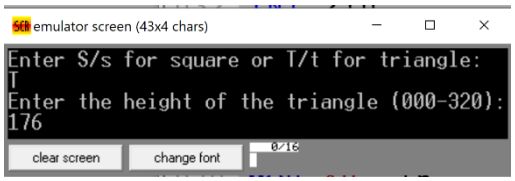


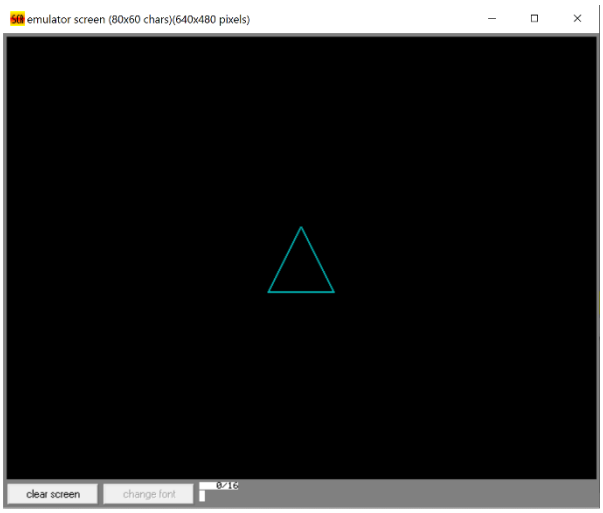


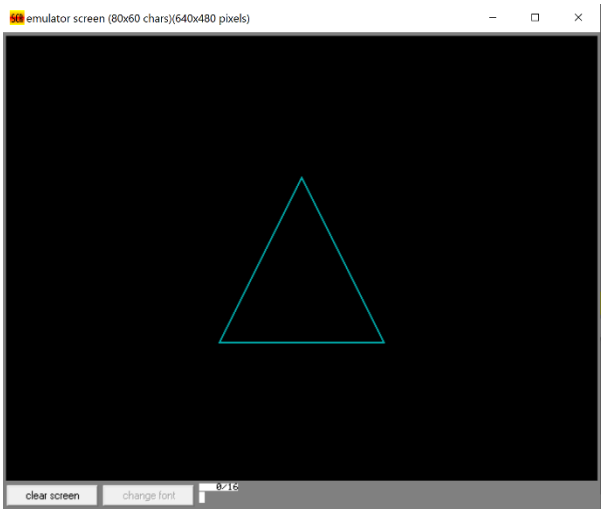


Triangle Examples:

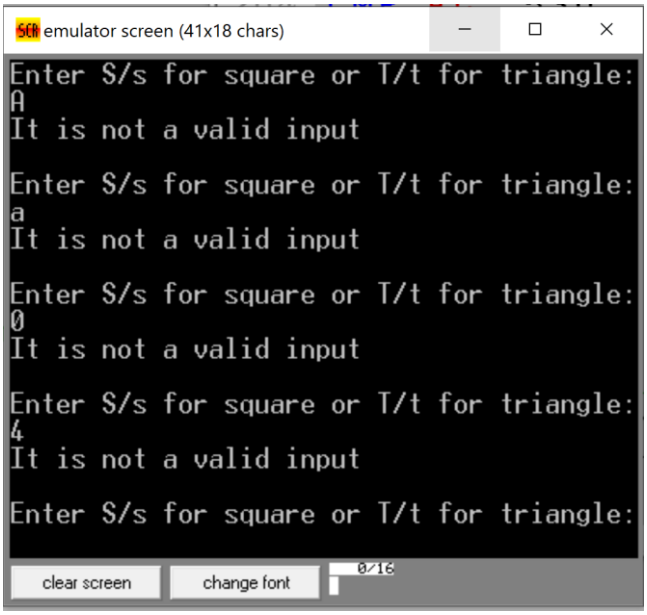


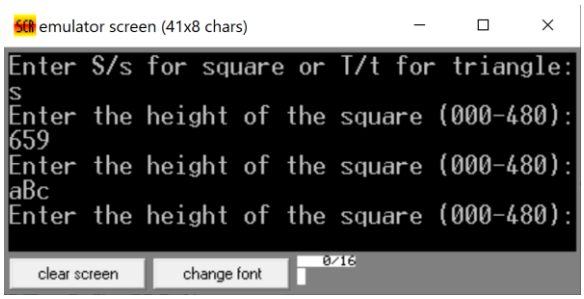


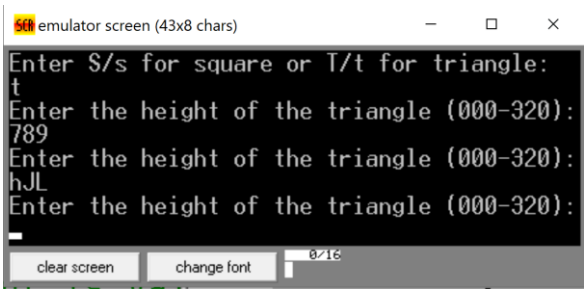




Invalid Case Examples:







Comments:

Firstly, program prompts user to enter a shape(a square or a triangle). If the entered input is not "S/s" or "T/t", program prints "It is not a valid input" and keeps repeating the input prompt until user enters a valid input.

And then, program prompts user to enter a height value, this prompt keeps popping on the screen until a valid input is entered. The valid input for square is between (000-480), the values above 480 doesn't make sense because it wouldn't be possible to print that square on a 640x480 screen. And for triangle it is between (000-320), the values above 320 doesn't make sense either because we wouldn't be able to print them.

The user has to enter the height value in 3 digits because on the backend side, program converts that 3-digit input to a valid integer value that we can work on. This process takes place in **"formatInput"** part of the code. This procedure takes the input character by character and then it stores this input in **BP** as an integer.

After a valid height value is entered, the algorithm starts printing the desired shape with the given height value by making use of pixel operations of 8086 assembly.

In both cases(square and triangle), procedures calculate the starting and ending coordinates for every edge that is going to be printed. Starting point is stored in **(CX, DX)** and the ending point is stored in **(SI, DI)** as in **(X, Y)** format. This coordinates are updated accordingly before printing each side.

After all the sides of the desires shape are printed, the program terminates itself.

Q3)

Source Code:

```
org 100h

;ELE338 - Preliminary Work 4 - Question 3

;Anıl Karaca - 21728405

JMP start

msgShape: DB "Enter S/s for square or T/t for triangle:",0Dh,0Ah,24h

msgHeightSquare: DB "Enter the height of the square (000-480):",0Dh,0Ah,24h

msgHeightTriangle: DB "Enter the height of the triangle (000-320):",0Dh,0Ah,24h

msgError: DB "It is not a valid input",0Dh,0Ah,24h



;(triTopX, triTopY) stores the coordinates of the top edge of the printed triangle

triTopX DW ?

triTopY DW ?

;(triRightX, triRightY) stores the coordinates of the right edge of the printed triangle

triRightX DW ?

triRightY DW ?

;(triLeftX, triLeftY) stores the coordinates of the left edge of the printed triangle

triLeftX DW ?

triLeftY DW ?



;Prints new line on the emulator screen

printNewline PROC

    MOV DX, 0Dh

    MOV AH, 2h

    INT 21h

    MOV DX, 0Ah

    MOV AH, 2h

    INT 21h

RET

printNewline ENDP



;This procedure takes 3 digit input and stores it in BP

formatInput PROC

    ;1st digit input

    MOV AH, 1h

    INT 21h

    SUB AX, 30h

    MOV BL, 100d

    MUL BL

    MOV BP, AX

    ;2nd digit input

    MOV AH, 1h

    INT 21h

    SUB AX, 30h

    MOV BL, 10d

    MUL BL

    ADD BP,AX

    ;3rd digit input

    MOV AH, 1h

    INT 21h



    MOV AH, 0d

    SUB AX, 30h

    ADD BP, AX

RET

formatInput ENDP



squareProc PROC

    ;Prompt height of the square

    promptHeightSquare:

    CALL printNewline

    MOV DX, msgHeightSquare

    MOV AH, 9h

    INT 21h

    CALL formatInput

    CMP BP, 480d ;Check if the input is valid

    JGE promptHeightSquare ;Keep looping until the input is valid



    ;Find starting point CX, DX

    MOV BX, 2d

    MOV DX, 0d

    MOV AX, BP

    DIV BX



    MOV CX, 320d ;Set CX to be half of screen width

    MOV DX, 240d ;Set DX to be half of screen height



    SUB CX, AX

    SUB DX, AX
```

```
;Find ending point SI, DI
MOV SI, CX
ADD SI, BP

MOV DI, DX
ADD DI, BP

;Open 640x480 screen
MOV AH, 0d
MOV AL, 12h
INT 10h
INT 10h

MOV AL, 3d ;Set the color of the pixels

;Print upper side
upperSide:
MOV AH, 0Ch
INT 10h
INC CX ;Update coordinate
CMP CX, SI
JLE upperSide

;Print right side
rightSide:
MOV AH, 0Ch
INT 10h
INC DX ;Update coordinate
CMP DX, DI
JLE rightSide

;Print bottom side
SUB SI, BP
bottomSide:
MOV AH, 0Ch
INT 10h
DEC CX ;Update coordinate
CMP CX, SI
JGE bottomSide
```

```
    ;Print left side
    SUB DI, BP
    leftSide:
    MOV AH, 0Ch
    INT 10h
    DEC DX ;Update coordinate
    CMP DX, DI
    JGE leftSide

    CALL detectSquare
RET
squareProc ENDP

detectSquare PROC
    mouseLoopSquare:
    ;(CX,DX) in (x,y) format
    ;BX=1 -> left mouse button down
    MOV AX, 3d
    INT 33h
    CMP BX, 1d ;Check if the left mouse button is clicked
    JNZ mouseLoopSquare

    isClickDone:
    MOV AX, 3d
    INT 33h
    CMP BX, 0d
    JNZ isClickDone

    ;Check the left side in terms of X coordinates
    checkX1:
    SHR CX, 1 ;Divide CX by 2 to get the correct X coordinate
    CMP CX, SI
    JGE checkX2
    JMP mouseLoopSquare

    ;Check the right side in terms of X coordinates
    checkX2:
    ADD SI, BP
    CMP CX, SI
    JLE checkY1
    SUB SI, BP ;Restore the original X coordinate of the starting point
    JMP mouseLoopSquare
```

```
    ;Check the upper side in terms of Y coordinates
    checkY1:
    SUB SI, BP ;Restore the original X coordinate of the starting point
    CMP DX, DI
    JGE checkY2
    JMP mouseLoopSquare

    ;Check the bottom side in terms of Y coordinates
    checkY2:
    ADD DI, BP
    CMP DX, DI
    JLE exit
    SUB DI, BP ;Restore the original Y coordinate of the starting point
    JMP mouseLoopSquare

exit:
RET
detectSquare ENDP

detectTriangle PROC
    ;Store the X coordinate of the middle of the left side in AX
    MOV AX, triTopX
    ADD AX, triLeftX
    SHR AX, 1 ;Divide AX by 2 to get the right coordinate

    ;Store the Y coordinate of the middle of the left side in BX
    MOV BX, triTopY
    ADD BX, triLeftY
    SHR BX, 1 ;Divide BX by 2 to get the right coordinate

    ;Store the X coordinate of the middle of the right side in CX
    MOV CX, triTopX
    ADD CX, triRightX
    SHR CX, 1 ;Divide CX by 2 to get the right coordinate

    ;Store the Y coordinate of the middle of the right side in DX
    MOV DX, triTopY
    ADD DX, triRightY
    SHR DX, 1 ;Divide DX by 2 to get the right coordinate

    ;Store the height of the imaginary square in BP
    SUB CX, AX
    MOV BP, CX

    ;Store the coordinates of the upper left edge of the imaginary
square
    ;(AX,BX)->(SI,DI)
    MOV SI, AX
    MOV DI, BX

    ;Call detectSquare for the imaginary square
    CALL detectSquare

RET
detectTriangle ENDP

triangleProc PROC
    ;Prompt height of the triangle
    promptHeightTriangle:
    CALL printNewline
    MOV DX, msgHeightTriangle
    MOV AH, 9h
    INT 21h
    CALL formatInput
    CMP BP, 320d ;Check if the input is valid
    JGE promptHeightTriangle ;Keep looping until the input is valid

    ;Find starting point CX, DX
    MOV BX, 2d
    MOV DX, 0d
    MOV AX, BP
    DIV BX
    PUSH AX ;Store this value so we can use it later

    MOV CX, 320d ;Set CX to be half of screen width
    MOV DX, 240d ;Set DX to be half of screen height

    SUB DX, AX

    ;Store the top edge coordinates
    MOV triTopX, CX
    MOV triTopY, DX
```

```
;Find ending point SI, DI
MOV SI, CX
ADD SI, AX

MOV DI, DX
ADD DI, BP

;Store the right edge coordinates
MOV triRightX, SI
MOV triRightY, DI

;Open 640x480 screen
MOV AH, 0d
MOV AL, 12h
INT 10h
INT 10h

MOV AL, 3d ;Set the color of the pixels

;Print right side
rightSideTriangle:
MOV AH, 0Ch
INT 10h
ADD DX, 2d ;Update coordinate
INC CX
CMP DX, DI
JLE rightSideTriangle

;Print bottom side
SUB SI, BP

;Store the left edge coordinates
MOV triLeftX, SI
MOV triLeftY, DI

bottomSideTriangle:
MOV AH, 0Ch
INT 10h
DEC CX ;Update coordinate
CMP CX, SI
JGE bottomSideTriangle
```

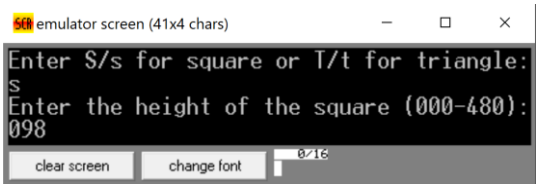
```
    ;Print left side
    POP AX
    ADD SI, AX
    SUB DI, BP
    MOV AL, 3d ;Set the color of the pixels again
    leftSideTriangle:
    MOV AH, 0Ch
    INT 10h
    SUB DX, 2d ;Update coordinate
    INC CX
    CMP DX, DI
    JGE leftSideTriangle

    CALL detectTriangle
RET
triangleProc ENDP

start:
;Print prompt msg to get the shape
MOV DX, msgShape
MOV AH, 9h
INT 21h
;Get the shape input
MOV AH, 1h
INT 21h

;Check if the input is "S" or "s"
CMP AL, 53h
JZ square
CMP AL, 73h
JZ square
;Check if the input is "T" or "t"
CMP AL, 54h
JZ triangle
CMP AL, 74h
JZ triangle

;If the input is invalid print error msg
CALL printNewline
MOV DX, msgError
MOV AH, 9h
INT 21h
CALL printNewline
JMP start
```

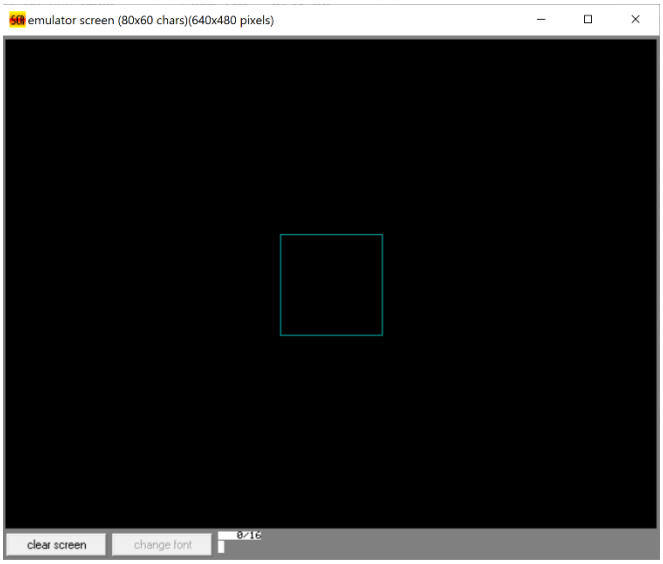
```
square:

CALL squareProc ;Print square by calling it's procedure

JMP terminate

triangle:

CALL triangleProc ;Print triangle by calling it's procedure

JMP terminate


terminate: ;Terminate the program

ret
```

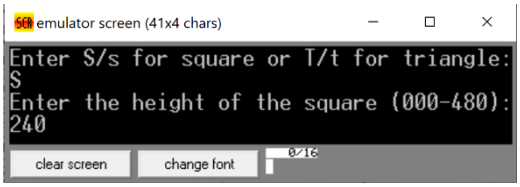
Screenshots:

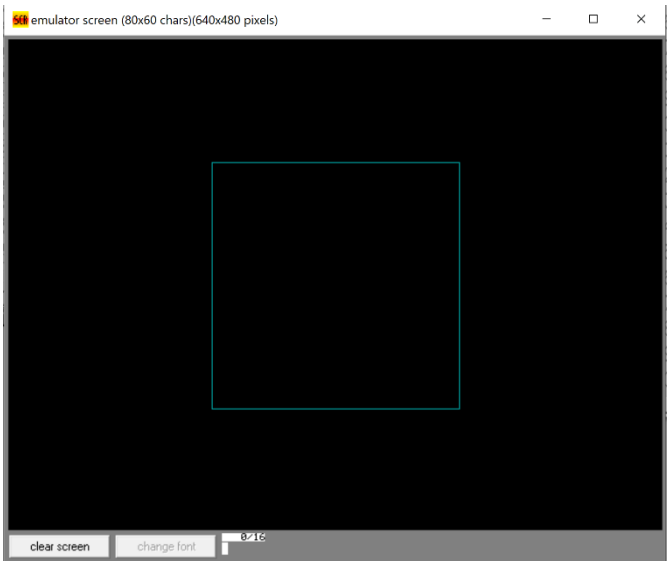
Emulator Screen:

Square Examples:

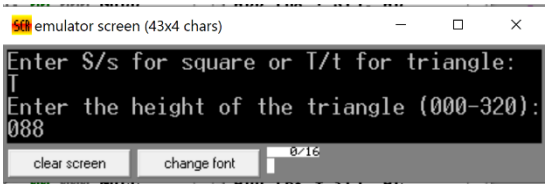


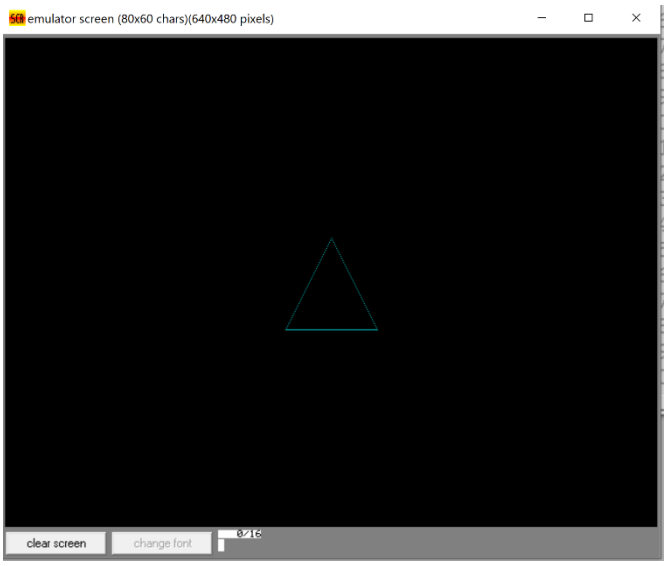


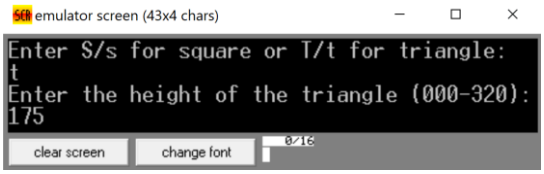


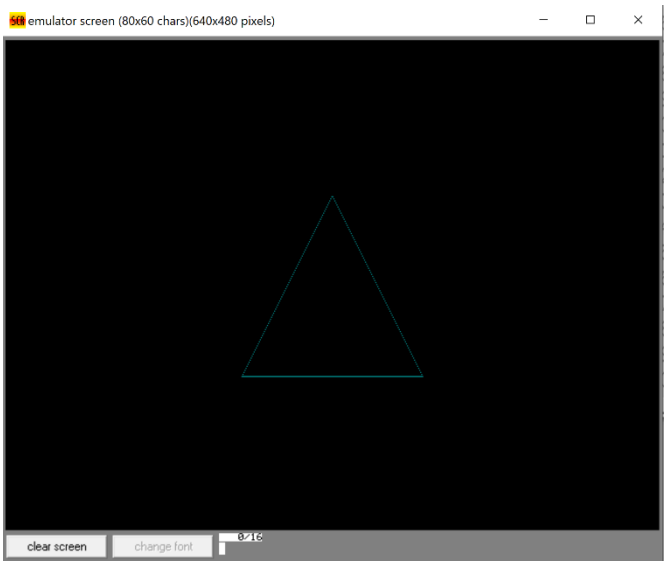


Triangle Examples:









Comments:

Everything I've talked about earlier on the **"Comments"** section of the 2nd question still remains the same. But for this question I had build a detection algorithm to analyze the mouse movements, clicks etc.

"squareProc" procedure in the code did not change at all, but for the detection of the mouse actions for square shapes I've coded a new subroutine called **"detectSquare"**. The **"mouseLoopSquare"** and **"isClickDone"** parts of this procedure, deals with the mouse actions. **"mouseLoopSquare"** loop directs the program to **"isClickDone"** when the left mouse button is pressed. And the, **"isClickDone"** redirects the program to the control mechanism when the left mouse button is elevated. And finally, the control mechanism checks the the location of the mouse cursor to indicate if the program should terminate or not.

This mechanism uses the coordinates of the upper left corner of the square **(SI, DI)**, and it compares this point with the coordinates of the mouse cursor **(CX, DX)**. My algorithm first eliminates according to the left side of the given square, then it eliminates the right side of it, and then it eliminates the upper side of that square and finally it eliminates the lower side of the square.

The fundamentals of the **"triangleProc"** did not change at all, but I've added some simple lines to store the coordinates of the edges of the printed triangle to be able to use them later on. These coordinates are stored in the variables called "triTopX", "triTopY", "triRightX", "triRightY", "triLeftX", "triLeftY". (triTopX, triTopY) stores the coordinates of the top edge, (triRightX, triRightY) stores the coordinates of the right edge and finally (triLeftX, triLeftY) stores the coordinates of the left edge.

By using this variables I calculate the coordinates of the upper left corner and the height of an imaginary square to be able to detect whether the point where the user has clicked was in the triangle or not. This process takes place in the **"detectTriangle"** procedure. **"detectTriangle"** calculates the middle point of the left side of the triangle and stores it's coordinates in **(AX, BX)**, and then it calculates the middle point of the right side of the printed triangle and stores it's coordinates in **(CX, DX)**. Later on, this subroutine calculates the height of this imaginary square by subtracting **AX** by **CX** then it stores this value in **BP**. And finally, it stores the coordinates of the upper left edge of this imaginary square in **(AX, BX)**->**(SI, DI)**.

Since now we know all the necessary details of this imaginary square, we can basically call the **"detectSquare"** procedure to detect the mouse actions like I've explained before.

Q4)

Source Code:

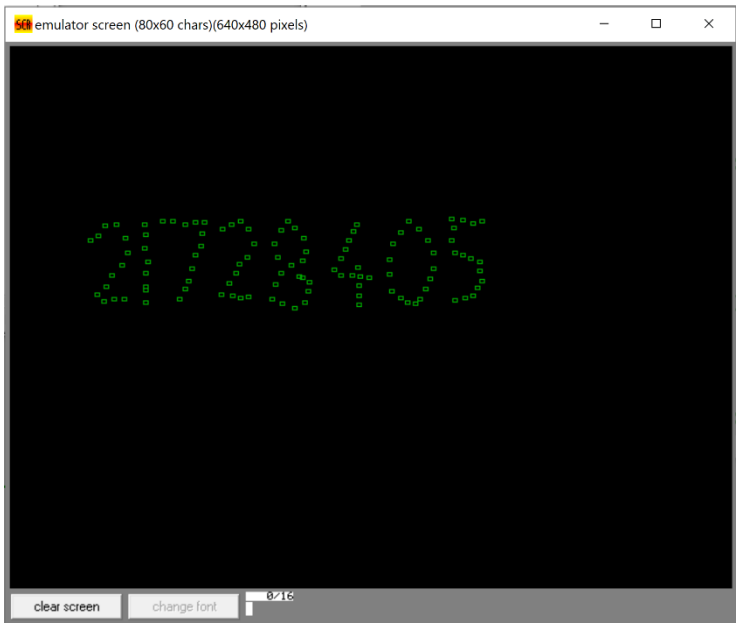
```
org 100h

;ELE338 - Preliminary Work 4 - Question 4

;Anıl Karaca - 21728405


JMP start


;Detect mouse cursor

detectMouse PROC

    mouseLoop:

    ;(CX,DX) in (x,y) format

    ;BX=1 -> left mouse button down

    MOV AX, 3d

    INT 33h

    CMP BX, 1d ;Check if the left mouse button is clicked

    JNZ mouseLoop

    CALL drawBox

    JMP mouseLoop


RET

detectMouse ENDP


;Draw a 2x2 box

drawBox PROC

    ;Store the location of the mouse cursor in (SI,DI) form

    SHR CX, 1 ;Divide CX by 2 to get the correct x-coordinate

    MOV SI, CX

    MOV DI, DX


    ;Set starting point(Upper Side)

    SUB CX, 1d

    SUB DX, 1d


    MOV AL, 2d ;Set the color of the pixels
```

```
    ;Set ending point(Upper Side)

    ADD SI, 1d

    ;Print upper side

    upperSide:

    MOV AH, 0Ch

    INT 10h

    INC CX ;Update coordinate

    CMP CX, SI

    JLE upperSide


    ;Set ending point(Right Side)

    ADD DI, 1d

    ;Print right side

    rightSide:

    MOV AH, 0Ch

    INT 10h

    INC DX ;Update coordinate

    CMP DX, DI

    JLE rightSide


    ;Set ending point(Bottom Side)

    SUB SI, 2d

    ;Print bottom side

    bottomSide:

    MOV AH, 0Ch

    INT 10h

    DEC CX ;Update coordinate

    CMP CX, SI

    JGE bottomSide
```

```
    ;Set ending point(Left Side)
    SUB DI, 2d
    ;Print left side
    leftSide:
    MOV AH, 0Ch
    INT 10h
    DEC DX ;Update coordinate
    CMP DX, DI
    JGE leftSide

RET
drawBox ENDP

start:
;Open 640x480 screen
MOV AH, 0d
MOV AL, 12h
INT 10h
INT 10h

CALL detectMouse ;Initiate the process
ret
```

Screenshots:

Emulator Screen:



Comments:

Firstly, the program opens the emulator screen in video-mode where the screen size is **(640x480).** And then, it calls the **"detectMouse"** procedure to detect the mouse movements and the clicks on the emulator screen. The "mouseLoop" part in the procedure allows us to keep detecting mouse movements until a left mouse button is clicked.

When the left mouse button is clicked, **"detectMouse"** procedure calls the **"drawBox"** subroutine, which allows us to draw a box shape on the screen where the left mouse button was clicked.

I've found printing a **(2x2)** box for each click to be neat and good looking, eventhough my algorithm is capable of printing a box with different shapes as well. I could have stored the height or the width of the box in a register but I didn't want my code to be overcomplicated.

The algorithm I've applied in **"drawBox"** subroutine is pretty straight forward and also very similar to my algorithm for the 2nd question of this preliminary work. The subroutine basically sets the starting and ending coordinates for each side, and then it prints pixels on the screen until we reach the ending coordinates.

And finally, when the desired box is drawn on the screen, **"drawBox"** procedure terminates and directs the program back to **"detectMouse"** subroutune to keep up the program by using the "mouseLoop" part.