SUDOKU MYSTIFIER

INSHA SAMNANI	**(20k-0247)**
ISMAIL AHMED ANSARI	**(20k-0228)**
YUSRA ADAM	**(20k-0207)**

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES - FAST

INTRODUCTION:

Sudoku, sounds interesting? A 9 by 9 grid that needs to be completed in such a way that each row, each column and each of the 9 main 3 by 3 sub-grids contains just one of each of the numbers 1 to 9. Our project “Sudoku Mystifier” also performs the same functionality. Solving the sudoku puzzle with user defined inputs as well as solving grid with random values detecting whether the inputs are valid or not is what “Sudoku Mystifier” does.

LITERATURE REVIEW:

Our project “Sudoku Mystifier”, fulfils the demand of solving the grid keeping in mind of all the rules. If any invalid input is being provided user is asked to enter again until rules are being fulfilled.

Solving a sudoku puzzle with accuracy is being done with the help of many algorithms. One algorithm is the “Paper-pencil” algorithm. In this algorithm human based used strategies are used to fulfill the requirements. The methods used in this algorithm are unique missing candidates, naked singles and hidden singles. Another algorithm used to solve sudoku puzzles is the “Brute Force” algorithm. In this algorithm, a random solution is being generated and after that it is checked whether the solution is correct or not. If it is not correct, then the process is repeated again. This algorithm can be time consuming because it will go through all possibilities of solutions. A scholar named “Nelishia Pillay” suggests solution for solving sudoku puzzles by combining human intuition and optimization.

Many algorithms have been considered to solve the sudoku puzzles but one thing every expert looks is optimized algorithm that produces accurate results.

PROBLEM DEFINATION:

Solving a sudoku puzzle with efficiency and accuracy can sometimes be something worth time consuming. Many algorithms have been previously used in order to generate accurate results after solving the sudoku grid. Online websites providing algorithms are not always worth to follow as there can be a chance of wrong implementation of rules. “Sudoku Mystifier” makes sure that the grid provided by user is solved keeping in view all the rules. Moreover, our project enables user to enter own values and if any invalid value is being entered, user is notified and asked to enter a new value.

METHADOLOGY/SOLUTION STATEMENT:

TOOLS USED ARE: Irvine32.inc Library, Microsoft Visual Studio Code 2019.

Following is the flow of our project. The details also includes different techniques and algorithms used in the respective functions.

- WelcomeInterface: The function is showing the front cover of our output (using setTextColor, writeString, clrscr). It is also taking the option as input from user (0 for playing game, 1 for getting your Sudoku solved) through ReadInt.
- PlayGame: The function is further calling ReadTheGrid, PrintGrid, and inputValues.
    - ReadTheGrid: The function is using filing and its functions (SIZEOF, OFFSET, OpenInputFile, and ReadFromFile) to read one Sudoku puzzle out of different Sudoku puzzles from the file. It is using Randomize and RandomRange function for selection. (INC, CMP, and Jumps are also used).
    - PrintGrid: The function is further calling printHorizontalLine using different variables to display a user defined grid to play the game in a successful user interface. It will also make the selected Sudoku to be displayed on the console along with the grid. Jumps are used to achieve respective tasks. (CMP, writeString, writeDec etc are also used).
    - InputValues: The function is calling findEmptyPlace to see if any of the element in the grid is 0 so that it will continue the process of taking values as input from user. Then user will be asked about row no col no and a value to be placed. All these will be verified via function Authentication1. It is also calling checkGrid function to check if the required space is containing 0(empty). Then value along with row and column will be checked via function isvalidplace. IsValidPlace further calls three functions: isPresentInRow, isPresentInCol, isPresentInBox. All these functions check if the value entered by the user are not repeated in the specified row, column, and the box. (CMP, INVOKE,

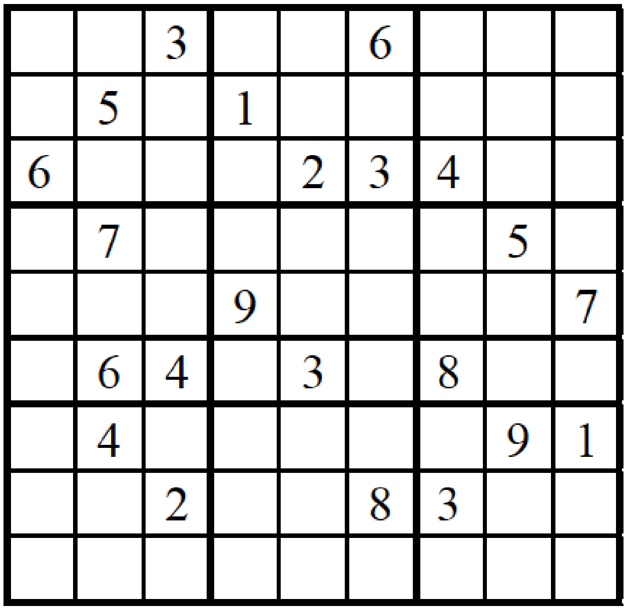
and Jumps are used). LEA is also used to hold offsets of row and columns. It is also calling PrintGrid function to print grid immediately after user enters the value again and again.

- Getting your Sudoku solved (sudokuSolver): Firstly the function is calling inputgrid function, solverSudoku function.
    - Inputgrid: The function calls PrintGrid function. It is using nested looping to take input of all the 9 values of 9 respective rows from the user. All the values are checked through Authentication0 function. (INC, DEC, clrscr, writeString, CMP etc are used).
    - SolveSudoku: It is a recursive approach to solve Sudoku for the user. It traverses whole grid and tries to place a value starting from 1 if it is valid via isvalidplace. If it finds valid so indexes of rows and columns are incremented respectively. Else it is backtracked and filled with 0 to again check it for incremented value that is 2. The process is continued unless it is unable to find some empty space (0) via findEmptyPlace. (LEA, CMP, PUSH, Jumps, INC etc are used).
- Now after the whole process, it is checking that whether the Sudoku is solved or not, if solved so printed via PrintGrid, else an error is shown that Sudoku can’t be solved.

DETAILED DESIGNED AND ARCHITECTURE:

cin >> option; to play game or get your Sudoku solved?

FOR PLAYING THE GAME:

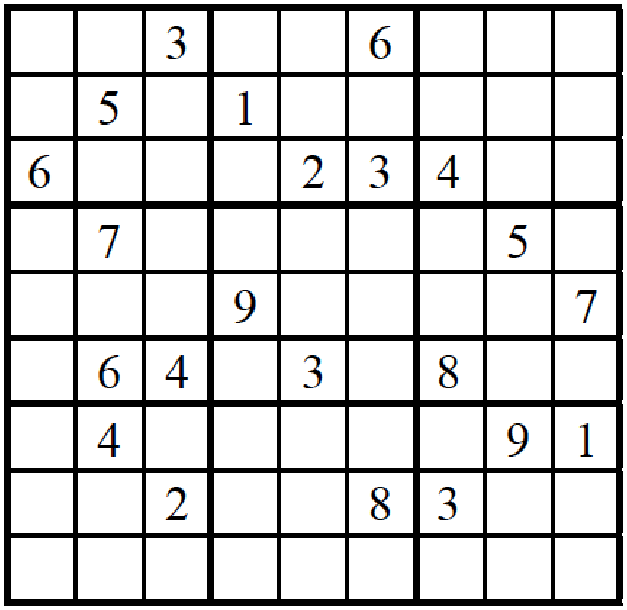


		3			6			
	5		1					
6				2	3	4		
	7						5	
			9					7
	6	4		3		8		
	4						9	1
		2			8	3		

NOTE: **(0s will be filled in empty spaces)**

```
While any of the element in the grid is 0
        Read the row number
        Authentication1(); value lie between 1-9
        Read the column number
        Authentication1(); value lie between 1-9
        checkGrid(); desired location is having 0?
        Read value from user; value lie between 1-9
        isValidPlace(); same value isn't present in that row, column or box
        PrintGrid(); print the grid everytime for the user after inserting a single value
```

FOR GETTING YOUR SUDOKU SOLVED:



		3			6			
	5		1					
6				2	3	4		
	7						5	
			9					7
	6	4		3		8		
	4						9	1
		2			8	3		

NOTE: (0s will be filled in empty spaces)

Inputgrid(); above mentioned incomplete Sudoku puzzle will be taken as input from the user along with 0s as spaces.

Authentication0(); all values entered by Inputgrid() lie between 0-9

solveSudoku(); recursive approach (including backtracking) to let computer analyze the appropriate value for the specific row and column one by one in the incomplete puzzle by respectively incrementing the value or indexes and starting from 1.

```
if(boolval==1); if user input all values according to the Sudoku rules
        PrintGrid(); printing solved Sudoku for user
else
        cout << msg; Sudoku can’t be solved
```

IMPLEMENTATION AND TESTING AND PROGRAM CODE:

```
INCLUDE Irvine32.inc
;SUDOKU SOLVER AND GAME

Authentication0 PROTO, valt:DWORD
Authentication1 PROTO, valt:DWORD
checkGrid PROTO, rowNo:DWORD, colNo:DWORD
.data
        ;GRID-INFO:
        N BYTE 9
        score DWORD 0
        grid BYTE 81 DUP (?)
        gridcopy BYTE 81 DUP (?)
        gridgame_tmp BYTE 1630 DUP(0)   ;81 ELEMENTS, 80 SPACES (EACH
LINE CONTAINS
        ;ONE SUDOKU-TOTAL 10 SUDOKUS)

        ;BOOLEANS:
        bool1 BYTE ?
        bool2 BYTE ?
        mbool3 BYTE 0
        bool4 BYTE ?
        bool5 BYTE 0
        bool6 BYTE 0

        ;MESSAGES:
```

```
	end_msg BYTE "\\----- GAME IS NOW EXITING -----//", 0
	mesg BYTE "The solved grid is: ", 0
	msg BYTE "No solution exists for the given grid!", 0
	msg2 BYTE "Unsolved Grid:", 0
	msg3 BYTE "Solved Grid:", 0
	msg4 BYTE "Enter the row no: ", 0
	msg5 BYTE "Enter the column no: ", 0
	msg6 BYTE "Enter the value: ", 0
	msg7 BYTE "You Won!!!", 0
	space BYTE ' ', 0
	multiplespace BYTE "   ", 0
	vertical BYTE "| ", 0
	horizontal BYTE "---", 0
	rowmsg BYTE "Enter the elements for ROW-", 0
	msgg BYTE "This is an INVALID VALUE! ", 0
	st_msg1 BYTE "Hey! I'm Shaggy!", 0
	st_msg2 BYTE "Do You Wanna PLAY SUDOKU with me or LET me SOLVE one with you??", 0
	st_msg3 BYTE "ENTER 0 TO START THE GAME", 0
	st_msg4 BYTE "ENTER 1 TO GET YOUR SUDOKU SOLVED", 0
	st_msg5 BYTE "Your Option: ", 0
	st_msg6 BYTE "ENTER ANY KEY TO GO BACK TO THE MAIN MENU......", 0
	sc_msg BYTE "Woah! Your score is ", 0
	;FILE-HANDLING:
	filename BYTE "Sudoku.txt", 0
	filehandle DWORD ?
```

```
.code
    Authentication1 PROC USES edx, valt:DWORD        ;checks whether the data is from 1 to 9
        MOV bool6, 0
        MOV edx, valt
        CMP edx, 9
        JNC J1
        CMP edx, 1
        JNC J2

        J1:
        JZ J2
        RET

        J2:
        MOV bool6, 1 ;TRUE
        RET
    Authentication1 ENDP

Authentication0 PROC USES edx eax, valt:DWORD      ;checks whether the data is from 0 to 9
        MOV bool5, 0
        MOV edx, valt
        CMP edx, 9
        JNC J1
        CMP edx, 0
        JNC J2
```

```
	J1:
	JZ J2
	RET

	J2:
	MOV bool5, 1 ;TRUE
	RET
Authentication0 ENDP

checkGrid PROC USES eax ebx esi, rowNo:DWORD, colNo:DWORD
	MOV bool6, 0
	MOV ebx, OFFSET gridcopy
	MOV eax, rowNo
	MUL N
	ADD ebx, eax
	MOV esi, colNo

	cmp BYTE PTR [ebx+esi], 0
	JNZ J1
		MOV bool6, 1
	J1:
	RET
checkGrid endp

inputValues PROC USES eax esi edx ebx
	LOCAL rowNo:DWORD, colNo:DWORD, Value:DWORD
```

```
L1:                     ; While Loop
        MOV rowNo, 0
        MOV colNo, 0
        LEA eax, rowNo
        LEA ebx, colNo

        PUSH eax
        PUSH ebx
        CALL findEmptyPlace
        ADD esp, 8

        CMP bool2, 1 ;empty place found
        JNZ J3

        MOV ebx, OFFSET grid
        MOV esi, 0
        MOV edx, OFFSET msg4
        CALL WriteString
        CALL readDec
        MOV rowNo, eax
        INVOKE Authentication1, rowNo
        CMP bool6, 1
        JNZ J1
        DEC rowNo

        MOV bool5, 0
MOV bool6, 0
MOV edx, OFFSET msg5
CALL WriteString
CALL readDec
MOV colNo, eax
INVOKE Authentication1, colNo
CMP bool6, 1
JNZ J1
DEC colNo

PUSH rowNo
PUSH colNo
INVOKE checkGrid, rowNo, colNo
CMP bool6, 1
JNZ J1

MOV bool6, 0
MOV bool5, 0
MOV edx, OFFSET msg6
CALL WriteString
CALL readDec
MOV Value, eax

MOV bool4, 0
PUSH rowNo
PUSH colNo
PUSH Value
```

```
CALL isvalidplace
ADD esp, 12
CMP bool4, 1
JNZ J1

MOV eax, rowNo
MUL N
ADD ebx, eax
MOV esi, colNo
MOV eax, Value
INVOKE Authentication1, eax
CMP bool6, 1
JNZ J1                          ;The case of invalid

ADD score, 5
MOV BYTE PTR [ebx+esi], al
CALL printGrid
JMP J2

J1:
CALL crlf
MOV edx, OFFSET msgg
CALL WriteString
CALL crlf
CALL crlf
CALL PrintGrid
```

```
        J2:
    JMP L1

    J3:
    MOV bool2, 0
    MOV edx, OFFSET msg7
    CALL crlf
    CALL writeString
    MOV edx, OFFSET sc_msg
    CALL crlf
    CALL crlf
    CALL crlf
    CALL WriteString
    MOV eax, score
    CALL WriteDec

    CALL crlf
    CALL crlf
    MOV edx, OFFSET st_msg6
    CALL WriteString
    CALL crlf
    CALL crlf

    CALL readChar

    RET
inputValues ENDP
```

```
inputgrid PROC USES eax ebx ecx edx
        LOCAL var_counter: DWORD
        MOV var_counter, 0
        MOVZX ecx, N
        MOV ebx, OFFSET grid
        MOV eax, 0
        L1:
                PUSH ecx
                MOV esi, 0
                CALL PrintGrid
                MOV edx, OFFSET rowmsg
                CALL Writestring
                MOV eax, var_counter
                INC eax
                CALL WriteDec
                MOVZX ecx, N
                CALL crlf
                L2:
                        CALL ReadInt
                        INVOKE Authentication0, eax
                        CMP bool5, 1
                        JNZ JM2

                        CMP eax, 0
                        JZ J1
                        PUSH var_counter
                        PUSH esi
                        PUSH eax
                        CALL isvalidplace
                        ADD esp, 12
                        CMP bool4, 1
                        JNZ JM2
                        MOV bool4, 0
                        J1:
                        MOV BYTE PTR [ebx+esi], al
                        MOV eax, 0
                        INC esi             ;COL-INC
                        LOOP L2
                        JMP JM3
                        JM2:
                        MOV edx, OFFSET msgg
                        CALL WriteString
                        CALL crlf
                JMP L2
                JM3:
                INC var_counter
                CALL clrscr
                MOVZX edi, N
                ADD ebx, edi            ;ROW-JUMP
                POP ecx
                DEC ecx
                CMP ecx, 0
        JNZ L1
```

```
	RET
inputgrid ENDP

printHorizontalLine PROC USES ecx
	PUSH ebp
	MOV ebp, esp
	MOVZX ecx, N
	DEC ecx
	CALL crlf
	MOV edx, OFFSET multiplespace
	CALL WriteString
	L1:
		MOV edx, OFFSET horizontal
		CALL WriteString
	LOOP L1

	POP ebp
	RET
printHorizontalLine ENDP

PrintGrid PROC USES eax edx ebx esi edi ecx
	PUSH ebp
	MOV ebp, esp

	CALL printHorizontalLine
	CALL crlf
```

```
MOV ebx, OFFSET grid
MOVZX ecx, N

l1:
    MOV edi, ecx
    MOV esi, 0
    MOVZX ecx, N
    MOV edx, OFFSET multiplespace
    CALL WriteString
    l2:
        CMP ecx, 6
        JC J2
        JNZ J1
            MOV edx, OFFSET vertical
            CALL WriteString
        J2:
        CMP ecx, 3
        JNZ J1
            MOV edx, OFFSET vertical
            CALL WriteString
        J1:
        CMP ecx, 9
        JNZ J5
            MOV edx, OFFSET vertical
            CALL WriteString
        J5:
```

```
        MOVZX eax, BYTE PTR [ebx+esi]
        CALL WriteDec
        MOV edx, OFFSET space
        CALL WriteString
        INC esi

        CMP ecx, 1
        JNZ J6
                MOV edx, OFFSET vertical
                CALL WriteString
        J6:
LOOP l2
MOVZX eax, N
ADD ebx, eax

CMP edi, 7
JC J4
JNZ J3
CALL printHorizontalLine

J4:
CMP edi, 4
JNZ J3
CALL printHorizontalLine
J3:

CMP edi, 1
```

```
                JNZ J7
                CALL printHorizontalLine
                J7:

                CALL crlf
                MOV ecx, edi
                DEC ecx
                CMP ecx, 0
        JNZ l1
        POP ebp
        CALL crlf
        RET
PrintGrid ENDP

isPresentIncol PROC USES ecx esi ebx eax edx
        PUSH ebp
        MOV ebp, esp
        MOVZX ecx, N
        MOV ebx, OFFSET grid
        MOV esi, [ebp+32]    ;COL-VALUE
        MOVZX edx, BYTE PTR [ebp+28]  ;NUM-VALUE
        L1:
                PUSH ecx
                MOVZX ecx, N
                CMP BYTE PTR [ebx+esi], dl
                JZ J2
                ADD ebx, ecx            ;row-jump
        POP ecx
    LOOP L1

    POP ebp
    RET

    J2:
        POP ecx
        INC mbool3
        POP ebp
        RET
isPresentIncol ENDP

isPresentInrow PROC USES ecx esi ebx eax edx
    PUSH ebp
    MOV ebp, esp
    MOVZX ecx, N
    MOV ebx, OFFSET grid
    MOV eax, [ebp+32] ;row
    MUL N
    ADD ebx, eax
    MOV esi, 0     ;    COL-VALUE
    MOVZX edx, BYTE PTR [ebp+28]  ;NUM-VALUE
    L1:
        PUSH ecx
        MOVZX ecx, N
        CMP BYTE PTR [ebx+esi], dl
        JZ J2
        INC esi                         ;COL-INC
        POP ecx
    LOOP L1

    POP ebp
    RET

    J2:
        POP ecx
        INC mbool3
        POP ebp
        RET
isPresentInrow ENDP

isPresentInBox PROC USES ecx esi ebx eax edx edi
    PUSH ebp
    MOV ebp, esp
    MOV ebx, OFFSET grid
    MOV esi, 0

    MOVZX eax, BYTE PTR [ebp+40]            ;row-row%3
    MOVZX edi, BYTE PTR [ebp+32]            ;EDI = NUM
    MOV ecx,  3
    MUL N                                                       ;eax
    MOV edx, [ebp+36]                               ;col-col%3
    ADD ebx, eax
```

```
L1:
        PUSH ecx
        MOV ecx, 3
        MOV esi, edx
        L2:
                PUSH ecx
                MOVZX ecx, BYTE PTR [ebx+esi]
                CMP edi, ecx
                JZ J1
                POP ecx
                INC esi
        loop L2
        PUSH edx
        MOVZX edx, N
        ADD ebx, edx            ;OFFSET+9 (For next row)
        POP edx
        POP ecx
loop L1

POP ebp
RET

J1:
        INC mbool3
        POP ecx
        POP ecx
```

```
		POP ebp
	RET
isPresentInBox ENDP

isValidPlace PROC
	PUSH eax
	PUSH edx
	PUSH esi
	PUSH ebp
	MOV ebp, esp
	mov mbool3, 0

	PUSH DWORD PTR [ebp+28] ;ROW
	PUSH DWORD PTR [ebp+20] ;NUM
	call isPresentInRow
	add esp, 8

	PUSH [ebp+24] ;COL
	PUSH [ebp+20] ;NUM
	call isPresentInCol
	add esp, 8

	MOV esi, 3
	MOV edx, 0
	MOV eax, [ebp+28]
	DIV esi
	MOV eax, [ebp+28]
```

```
    SUB eax, edx
    PUSH eax                            ;row- (row%3)

    mov edx, 0
    mov eax, [ebp+24]
    DIV esi
    MOV eax, [ebp+24]
    SUB eax, edx

    PUSH eax                            ;col- (col%3)
    PUSH [ebp+20]                       ;NUM
    CALL isPresentInBox
    ADD esp, 12

    CMP mbool3, 0
    JNZ J1
    MOV bool4, 1
    J1:
    POP ebp
    POP esi
    POP edx
    POP eax
    RET
isValidPlace ENDP

findEmptyPlace PROC USES edx esi edi ebx eax ecx
;return address at ebp+28
```

```
PUSH ebp
MOV ebp, esp
MOV ebx, OFFSET grid

MOV eax, [ebp+36]               ;row-address
MOV edx, [ebp+32]               ;col-address
MOV DWORD PTR [eax], 0 ;row
MOV DWORD PTR [edx], 0  ;col

MOVZX ecx, N
L1:
        PUSH ecx
        MOV esi, 0

        MOV BYTE PTR [edx], 0

        MOVZX ecx, N
        L2:
                CMP BYTE PTR [ebx+esi], 0
                JZ ST1
                INC esi
                INC BYTE PTR [edx]
        LOOP L2
        MOVZX edi, N
        ADD ebx, edi
        INC BYTE PTR [eax]
        POP ecx
```

```
    LOOP L1
    POP ebp
    MOV bool2, 0 ;FALSE
    RET

    ST1:
    POP ecx
    POP ebp
    MOV bool2, 1 ;TRUE
    RET
findEmptyPlace ENDP

SolveSudoku PROC USES eax ebx esi edi edx
    PUSH ebp
    mov ebp, esp
    sub esp, 12
    lea eax, [ebp-4] ; row
    lea ebx, [ebp-8] ; col

    PUSH eax
    PUSH ebx
    CALL findEmptyPlace
    ADD esp, 8

    CMP bool2, 1 ;empty place found
    JZ J1
    MOV bool1, 1 ;TRUE
MOV esp, ebp
POP ebp
RET

J1:
MOV bool2, 0
MOV DWORD PTR [ebp-12], 1 ;try each number
MOVZX ecx, N
L1:
        PUSH [eax]    ;row
        PUSH [ebx]    ;col
        PUSH DWORD PTR [ebp-12]
        CALL isValidPlace
        ADD esp, 12

        CMP bool4, 1
        JNZ ST1
        MOV bool4, 0
        PUSH eax
        MOV edi, OFFSET grid
        MOV eax, [eax]                     ;row
        MOV esi, [ebx]                     ;col
        MUL N
        ADD edi, eax
        MOV edx, DWORD PTR [ebp-12]
        MOV BYTE PTR [edi+esi], dl
        POP eax
        MOV bool1, 0
        PUSH ecx
        CALL SolveSudoku
        POP ecx
        CMP bool1, 1
        JZ ST2

        PUSH eax
        MOV edi, OFFSET grid
        MOV eax, [eax]                          ;row
        MOV esi, [ebx]                          ;col
        MUL N
        ADD edi, eax
        MOV BYTE PTR [edi+esi], 0
        POP eax

        ST1:
        INC DWORD PTR [ebp-12]
        mov edi, DWORD PTR [ebp-12]
LOOP L1

mov esp, ebp
POP ebp
MOV bool1, 0 ;FALSE
RET
```

```
        ST2:
        mov esp, ebp
        POP ebp
        RET
    SolveSudoku ENDP

    ReadTheGrid PROC USES ecx edx eax esi edi ebx
        PUSH ebp
        MOV ebp, esp

        MOV score, 0
        MOV edx, OFFSET filename
        CALL OpenInputFile
        MOV filehandle, eax
        MOV ecx, SIZEOF gridgame_tmp
        MOV edx, OFFSET gridgame_tmp
        CALL ReadFromFile

        MOV ebx, OFFSET    gridgame_tmp
        MOVZX eax, N                            ;UPPER BOUND FOR PSEUDO-RANDOM NUMBER
        CALL Randomize
        CALL RandomRange                ;RANDOM NUMBER IN eax
        MOVZX ecx, N
        MOV edx, 0
        RANDGRID:
            CMP edx, eax
```

```
		JE ASS1
		ADD ebx, 163
		INC edx
LOOP RANDGRID

;transfering the values of the randomly selected grid without spaces
ASS1:
MOV esi, ebx
MOV ecx, LENGTHOF grid
MOV edi, OFFSET grid
CLD
L2:
		MOVSB
		INC esi
LOOP L2

MOV esi, OFFSET grid

MOV ecx, LENGTHOF gridcopy
MOV edi, OFFSET gridcopy
CLD
L3:
		MOVSB
LOOP L3

MOV ecx, LENGTHOF grid
MOV esi, OFFSET grid
```

```
        MOV edi, OFFSET gridcopy
        L1:
                SUB BYTE PTR [esi], 48
                INC esi
                SUB BYTE PTR [edi], 48
                INC edi
        LOOP L1

        POP ebp
        RET
ReadTheGrid ENDP

PlayGame PROC
        ENTER 1, 0
        CALL ReadTheGrid
        CALL PrintGrid
        CALL inputValues

        LEAVE
        RET
PlayGame ENDP

initialisegrid PROC USES ecx edi eax
        PUSH ebp
        MOV eax, 0
        MOV ecx, SIZE grid
        MOV edi, OFFSET grid
        CLD
        REP STOSB
        POP ebp
        RET
initialisegrid ENDP

SudokuSolver PROC USES edx
        PUSH ebp
        MOV ebp, esp

        CALL inputgrid
        CALL SolveSudoku
        CMP bool1, 1
        JNZ ST2
        MOV edx, OFFSET mesg
        CALL WriteString
        CALL crlf
        CALL crlf
        CALL PrintGrid
        JMP ST3

        ST2:
                MOV edx, OFFSET msg
                CALL WriteString
                CALL crlf
                CALL crlf
        ST3:
        CALL crlf
        CALL crlf
        MOV edx, OFFSET st_msg6
        CALL WriteString
        CALL crlf
        CALL crlf

        CALL readChar

        POP ebp
        RET
SudokuSolver ENDP

WelcomeInterface PROC USES ebx edx
        PUSH ebp
        MOV ebp, esp
        MOV eax, 16
        MOV bl, 15
        MUL bl
        CALL SetTextColor
        CALL clrscr

        MOV edx, OFFSET st_msg1
        CALL WriteString
        CALL crlf
```

```
	MOV edx, OFFSET st_msg2
	CALL WriteString
	CALL crlf

	MOV edx, OFFSET st_msg3
	CALL WriteString
	CALL crlf

	MOV edx, OFFSET st_msg4
	CALL WriteString
	CALL crlf

	MOV edx, OFFSET st_msg5
	CALL WriteString
	MOV eax, 0
	CALL ReadInt
	CALL clrscr

	POP ebp
	RET
WelcomeInterface ENDP



;ENTERING-POINT:
main PROC USES eax ebx edx ecx
	ENTER 0, 0
	L1:
        CALL WelcomeInterface
        ;USER'S CHOICE IS RETURNING IN EAX
        CMP eax, 0
        JNZ J1

        ;MODULE FOR THE SUDOKU-PLAYER (SUDOKU-GAME)
        CALL PlayGame
        JMP J2

        J1:
        CMP eax, 1
        JNZ J2

        ;MODULE FOR USER-GIVEN SUDOKU (SUDOKU-SOLVER)
        CALL initialisegrid
        CALL SudokuSolver

    Loop L1
    J2:
    CALL crlf
    CALL crlf
    MOV edx, OFFSET end_msg
    CALL crlf
    CALL WriteString
    CALL crlf
    LEAVE
```

```
                EXIT
                RET
        main ENDP
END MAIN
```

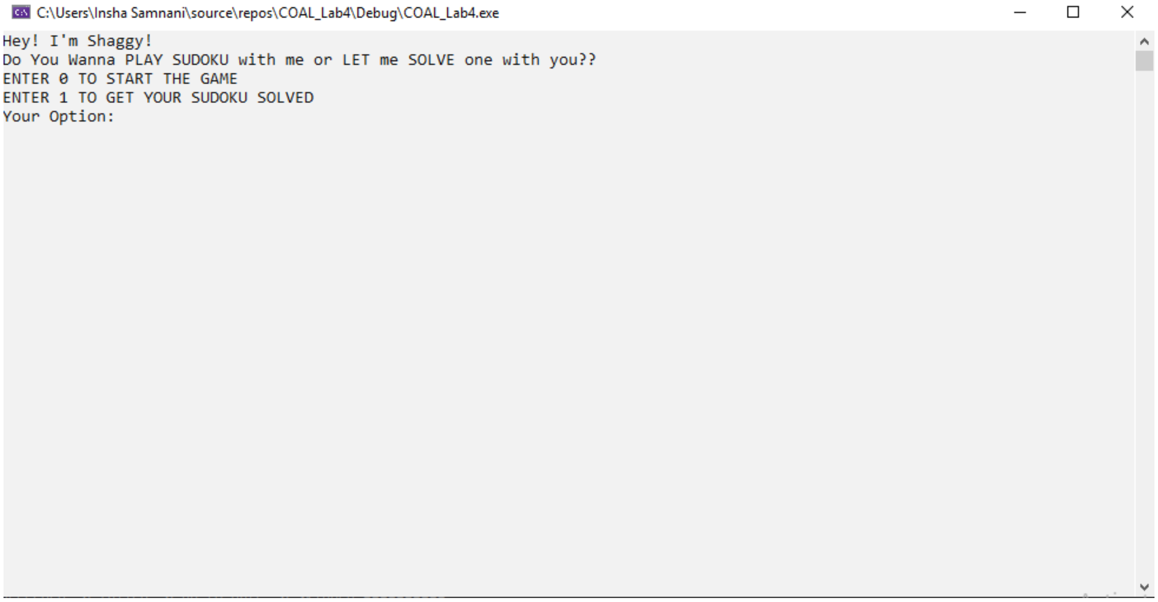
SUDOKU.TXT:

```
6 2 0 5 0 0 0 0 0 0 0 0 1 6 0 0 0 3 0 4 0 0 8 0 0 0 7 2 0 0 0 0 0 0 9 0 0 7 8 0 0 5 4 0 0 0
5 0 6 0 7 0 1 0 0 0 9 0 2 0 5 7 8 0 6 0 0 0 1 0 0 9 0 0 3 7 4 0 0 0 0

0 0 0 2 6 0 7 0 1 6 8 0 0 7 0 0 9 0 1 9 0 0 0 4 5 0 0 8 2 0 1 0 0 0 4 0 0 0 4 6 0 2 9 0 0 0
5 0 0 0 3 0 2 8 0 0 9 3 0 0 0 7 4 0 4 0 0 5 0 0 3 6 7 0 3 0 1 8 0 0 0

4 0 0 0 0 0 0 0 0 0 0 0 2 1 8 0 7 0 7 0 0 0 9 0 0 0 2 0 0 6 0 3 0 8 0 4 1 0 0 0 0 0 0 2 0 0
0 5 0 0 7 0 0 0 0 1 0 0 6 0 0 0 0 0 6 0 0 8 5 0 4 0 0 0 9 0 0 0 0 0 8

1 0 0 0 2 0 0 7 6 0 8 9 0 3 4 0 0 0 0 7 0 8 0 0 0 3 0 0 0 5 0 0 9 2 0 0 0 0 4 0 7 0 9 0 0 0
3 0 0 0 0 0 6 5 4 0 0 3 0 0 0 0 8 0 0 0 1 6 0 0 2 0 8 5 0 0 0 0 0 0 7

3 0 0 0 0 9 4 0 7 5 0 2 0 6 0 0 0 0 1 7 0 0 0 0 0 0 8 0 1 0 0 8 0 2 3 0 7 9 0 2 0 0 0 0 0 0
6 0 4 0 5 0 8 0 0 0 0 0 0 1 9 6 0 0 3 4 0 0 0 0 1 0 0 0 0 9 5 8 0 0 0

0 0 0 2 6 0 0 8 0 4 0 9 0 0 0 0 6 1 0 0 5 0 0 0 7 3 0 0 2 0 3 0 1 0 0 5 3 0 7 5 0 0 0 9 8 9
0 0 0 0 4 0 0 2 6 0 2 0 5 0 0 0 7 0 0 0 0 8 7 1 0 0 0 0 0 0 4 0 9 0 0

0 0 4 0 1 0 0 2 0 0 6 0 0 0 9 0 1 5 0 2 0 0 3 0 7 0 0 8 0 0 4 0 6 0 0 0 0 9 0 0 0 7 0 5 3 0
0 1 0 0 0 8 0 0 5 0 7 0 0 0 0 8 9 0 0 3 0 9 4 6 0 0 0 1 0 8 0 2 0 0 0

4 0 3 0 0 1 0 0 0 0 2 0 0 0 6 0 0 7 0 0 8 5 0 0 0 1 9 1 0 0 0 9 0 2 5 0 0 0 0 4 7 0 0 0 0 9
0 6 0 0 8 0 0 0 0 0 0 8 0 2 0 7 3 2 0 5 3 0 0 0 6 0 0 7 4 0 0 0 0 8 0

0 7 9 8 0 0 3 0 0 0 0 1 6 0 0 2 0 0 5 0 0 0 0 0 8 7 6 0 1 5 0 7 0 0 6 0 0 0 0 0 3 9 0 0 0 0
0 0 0 0 4 5 2 8 8 4 0 0 0 2 0 0 0 0 3 0 0 5 0 1 0 0 0 0 0 7 0 0 6 9 0
```

RESULTS, SOFTWARE SIMULATION AND DISCUSSION:

There are two options in our project:

1- Playing Game.
2- Getting Your Sudoku Solved.



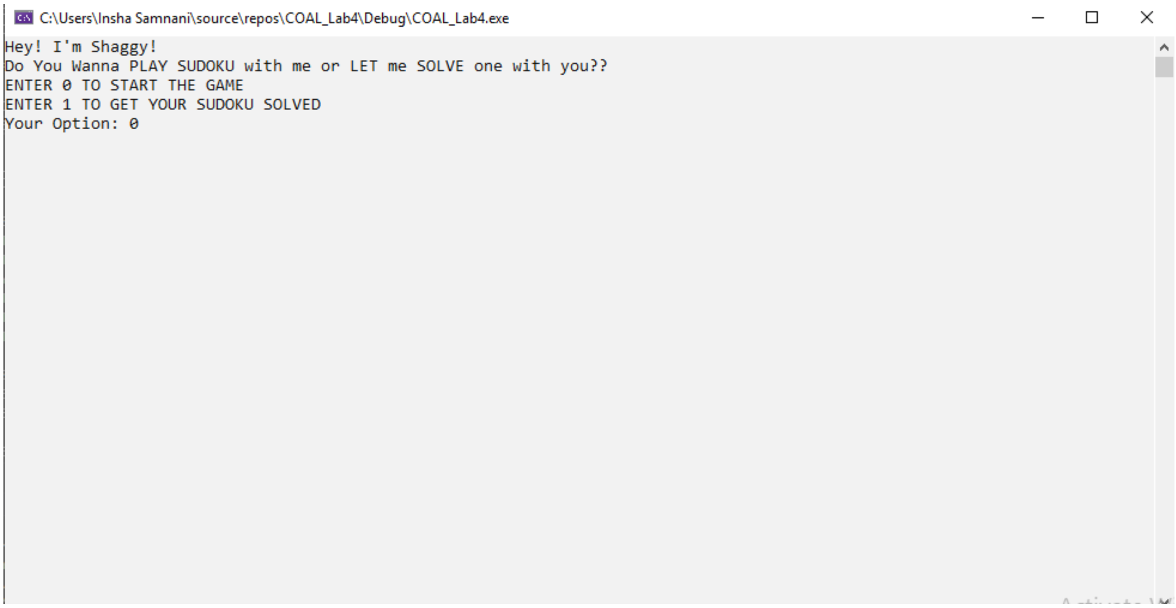
C:\Users\Insha Samnani\source\repos\COAL_Lab4\Debug\COAL_Lab4.exe

```
Hey! I'm Shaggy!
Do You Wanna PLAY SUDOKU with me or LET me SOLVE one with you??
ENTER 0 TO START THE GAME
ENTER 1 TO GET YOUR SUDOKU SOLVED
Your Option:
```

Option 1:

Test Case 1:

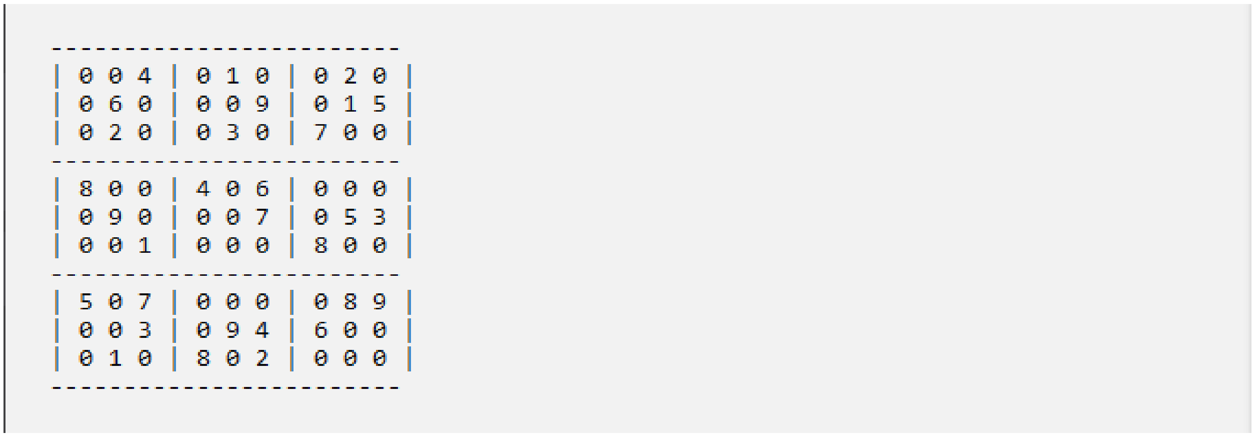
Outputs:



C:\Users\Insha Samnani\source\repos\COAL_Lab4\Debug\COAL_Lab4.exe

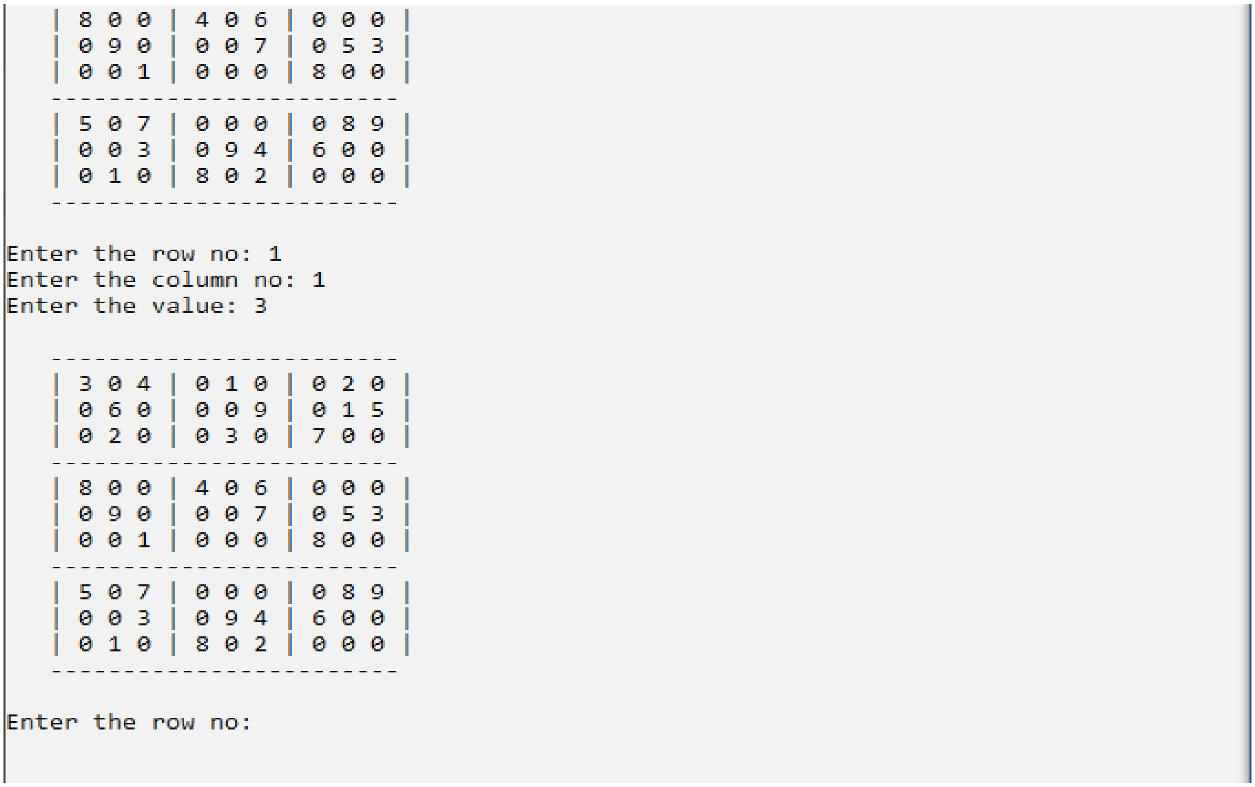
```
Hey! I'm Shaggy!
Do You Wanna PLAY SUDOKU with me or LET me SOLVE one with you??
ENTER 0 TO START THE GAME
ENTER 1 TO GET YOUR SUDOKU SOLVED
Your Option: 0
```

//Sudoku given to user to fill the missing(0's)



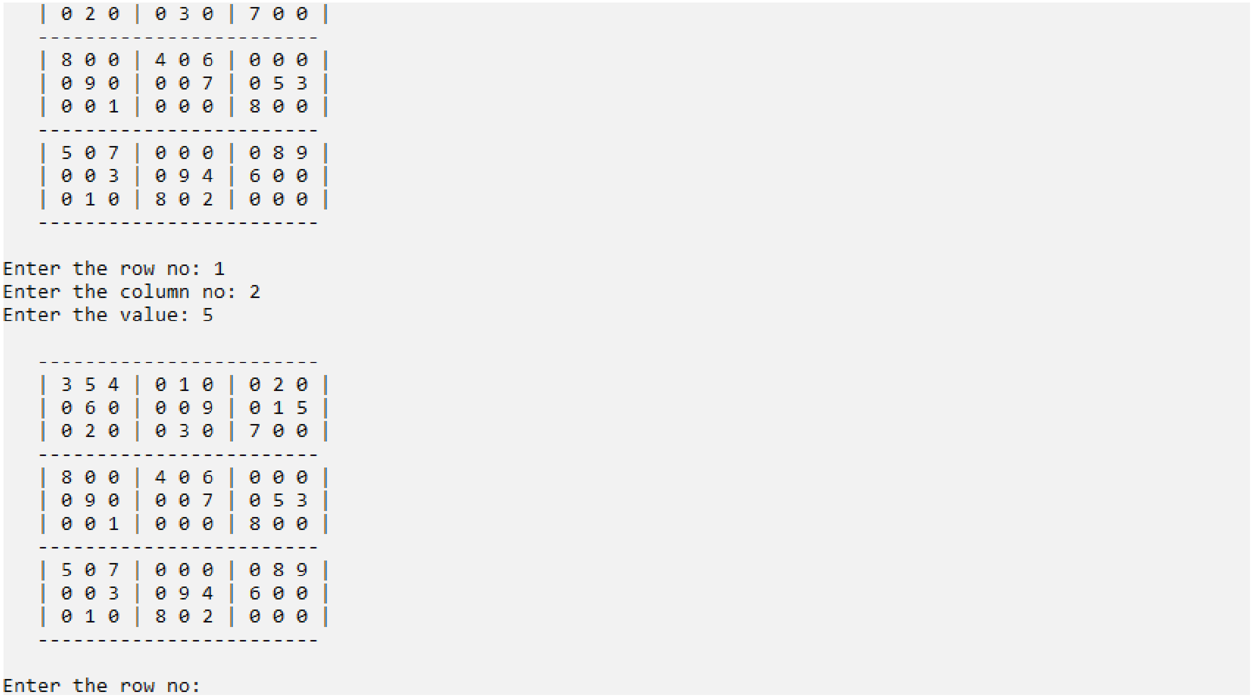
```
------------------------
| 0 0 4 | 0 1 0 | 0 2 0 |
| 0 6 0 | 0 0 9 | 0 1 5 |
| 0 2 0 | 0 3 0 | 7 0 0 |
------------------------
| 8 0 0 | 4 0 6 | 0 0 0 |
| 0 9 0 | 0 0 7 | 0 5 3 |
| 0 0 1 | 0 0 0 | 8 0 0 |
------------------------
| 5 0 7 | 0 0 0 | 0 8 9 |
| 0 0 3 | 0 9 4 | 6 0 0 |
| 0 1 0 | 8 0 2 | 0 0 0 |
------------------------
```

//Asks for row,col and the value to be inputted there   (1)



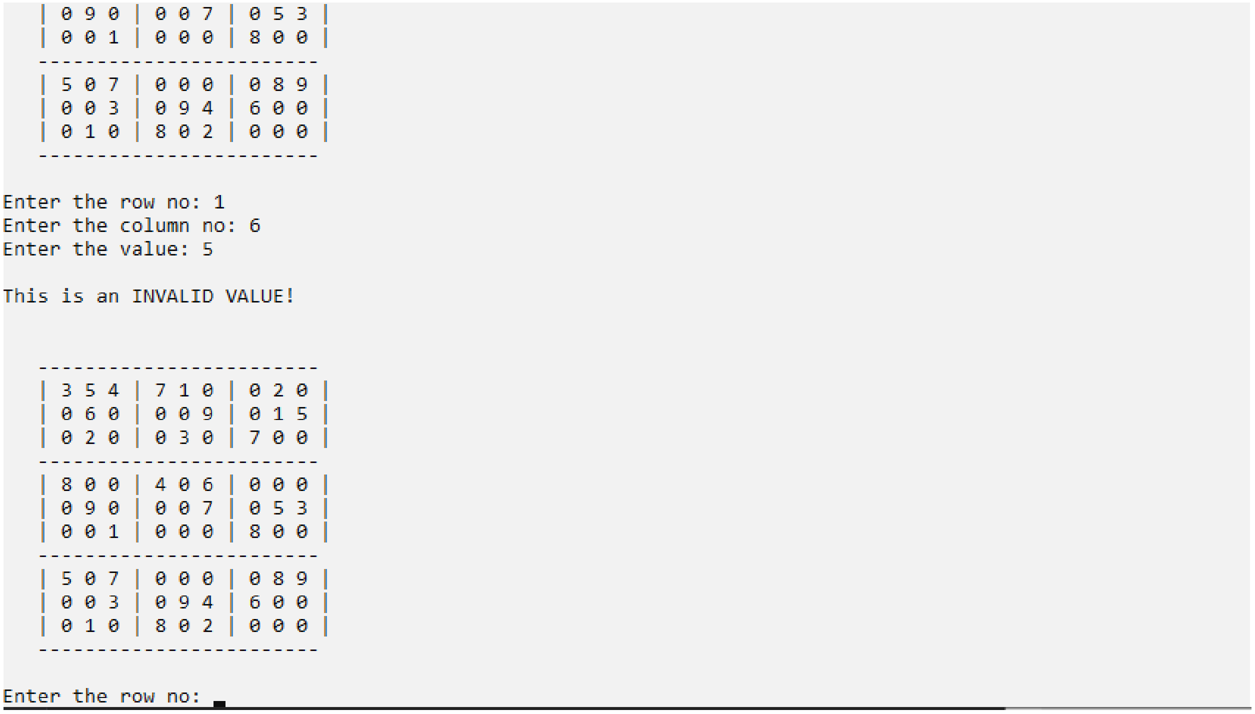
```
   | 8 0 0 | 4 0 6 | 0 0 0 |
   | 0 9 0 | 0 0 7 | 0 5 3 |
   | 0 0 1 | 0 0 0 | 8 0 0 |
   ------------------------
   | 5 0 7 | 0 0 0 | 0 8 9 |
   | 0 0 3 | 0 9 4 | 6 0 0 |
   | 0 1 0 | 8 0 2 | 0 0 0 |
   ------------------------

Enter the row no: 1
Enter the column no: 1
Enter the value: 3

   ------------------------
   | 3 0 4 | 0 1 0 | 0 2 0 |
   | 0 6 0 | 0 0 9 | 0 1 5 |
   | 0 2 0 | 0 3 0 | 7 0 0 |
   ------------------------
   | 8 0 0 | 4 0 6 | 0 0 0 |
   | 0 9 0 | 0 0 7 | 0 5 3 |
   | 0 0 1 | 0 0 0 | 8 0 0 |
   ------------------------
   | 5 0 7 | 0 0 0 | 0 8 9 |
   | 0 0 3 | 0 9 4 | 6 0 0 |
   | 0 1 0 | 8 0 2 | 0 0 0 |
   ------------------------

Enter the row no:
```

//Asks for row,col and the value to be inputted there   (2)



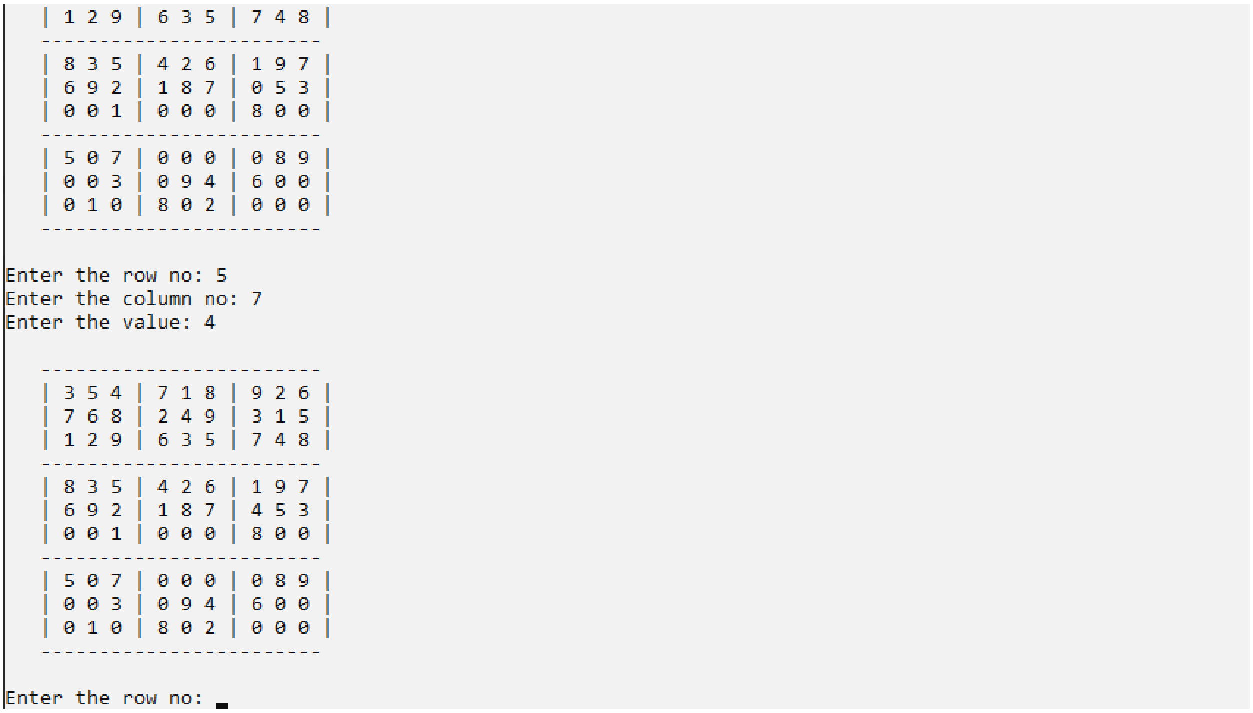
```
   | 0 2 0 | 0 3 0 | 7 0 0 |
   ------------------------
   | 8 0 0 | 4 0 6 | 0 0 0 |
   | 0 9 0 | 0 0 7 | 0 5 3 |
   | 0 0 1 | 0 0 0 | 8 0 0 |
   ------------------------
   | 5 0 7 | 0 0 0 | 0 8 9 |
   | 0 0 3 | 0 9 4 | 6 0 0 |
   | 0 1 0 | 8 0 2 | 0 0 0 |
   ------------------------

Enter the row no: 1
Enter the column no: 2
Enter the value: 5

   ------------------------
   | 3 5 4 | 0 1 0 | 0 2 0 |
   | 0 6 0 | 0 0 9 | 0 1 5 |
   | 0 2 0 | 0 3 0 | 7 0 0 |
   ------------------------
   | 8 0 0 | 4 0 6 | 0 0 0 |
   | 0 9 0 | 0 0 7 | 0 5 3 |
   | 0 0 1 | 0 0 0 | 8 0 0 |
   ------------------------
   | 5 0 7 | 0 0 0 | 0 8 9 |
   | 0 0 3 | 0 9 4 | 6 0 0 |
   | 0 1 0 | 8 0 2 | 0 0 0 |
   ------------------------

Enter the row no:
```

//For an Invalid Entry



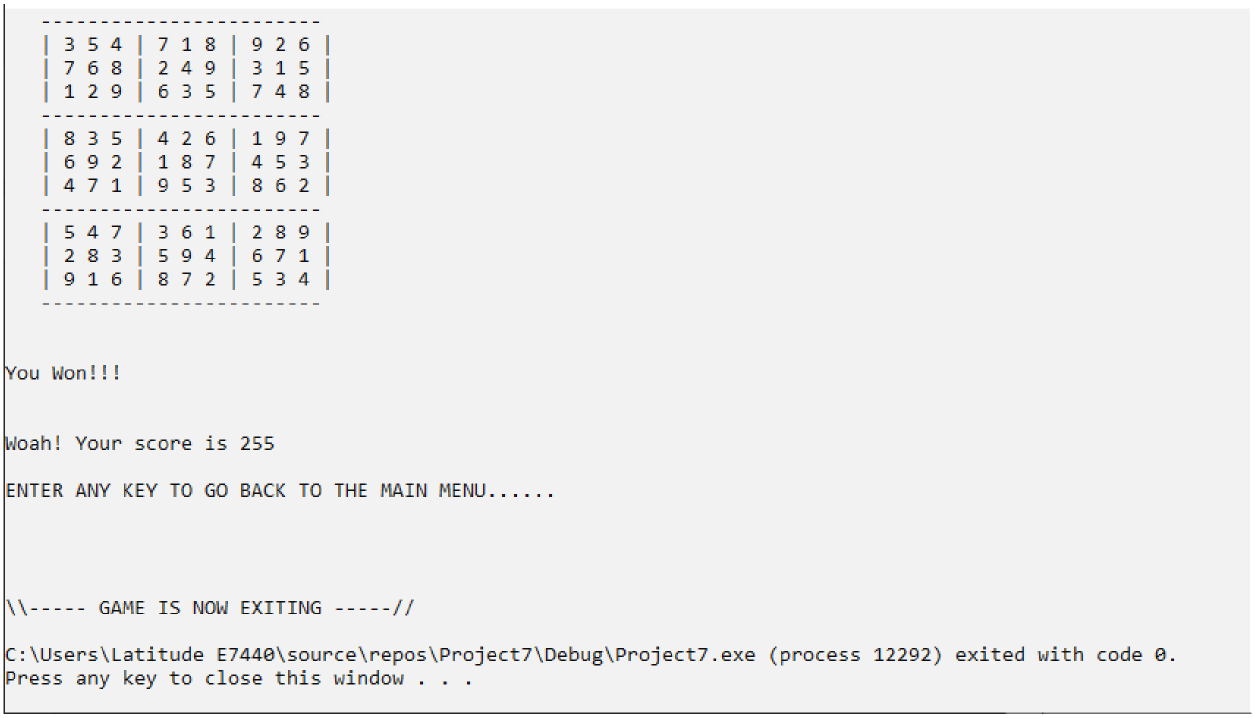
```
   | 0 9 0 | 0 0 7 | 0 5 3 |
   | 0 0 1 | 0 0 0 | 8 0 0 |
   ------------------------
   | 5 0 7 | 0 0 0 | 0 8 9 |
   | 0 0 3 | 0 9 4 | 6 0 0 |
   | 0 1 0 | 8 0 2 | 0 0 0 |
   ------------------------

Enter the row no: 1
Enter the column no: 6
Enter the value: 5

This is an INVALID VALUE!


   ------------------------
   | 3 5 4 | 7 1 0 | 0 2 0 |
   | 0 6 0 | 0 0 9 | 0 1 5 |
   | 0 2 0 | 0 3 0 | 7 0 0 |
   ------------------------
   | 8 0 0 | 4 0 6 | 0 0 0 |
   | 0 9 0 | 0 0 7 | 0 5 3 |
   | 0 0 1 | 0 0 0 | 8 0 0 |
   ------------------------
   | 5 0 7 | 0 0 0 | 0 8 9 |
   | 0 0 3 | 0 9 4 | 6 0 0 |
   | 0 1 0 | 8 0 2 | 0 0 0 |
   ------------------------

Enter the row no: 
```

//Intermediate Result



```
   | 1 2 9 | 6 3 5 | 7 4 8 |
   ------------------------
   | 8 3 5 | 4 2 6 | 1 9 7 |
   | 6 9 2 | 1 8 7 | 0 5 3 |
   | 0 0 1 | 0 0 0 | 8 0 0 |
   ------------------------
   | 5 0 7 | 0 0 0 | 0 8 9 |
   | 0 0 3 | 0 9 4 | 6 0 0 |
   | 0 1 0 | 8 0 2 | 0 0 0 |
   ------------------------

Enter the row no: 5
Enter the column no: 7
Enter the value: 4

   ------------------------
   | 3 5 4 | 7 1 8 | 9 2 6 |
   | 7 6 8 | 2 4 9 | 3 1 5 |
   | 1 2 9 | 6 3 5 | 7 4 8 |
   ------------------------
   | 8 3 5 | 4 2 6 | 1 9 7 |
   | 6 9 2 | 1 8 7 | 4 5 3 |
   | 0 0 1 | 0 0 0 | 8 0 0 |
   ------------------------
   | 5 0 7 | 0 0 0 | 0 8 9 |
   | 0 0 3 | 0 9 4 | 6 0 0 |
   | 0 1 0 | 8 0 2 | 0 0 0 |
   ------------------------

Enter the row no: 
```

//Final Result



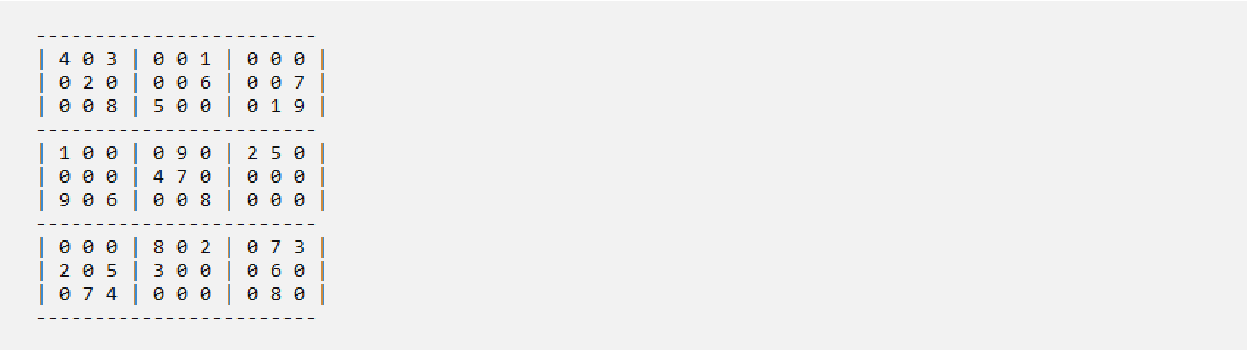
```
   ------------------------
   | 3 5 4 | 7 1 8 | 9 2 6 |
   | 7 6 8 | 2 4 9 | 3 1 5 |
   | 1 2 9 | 6 3 5 | 7 4 8 |
   ------------------------
   | 8 3 5 | 4 2 6 | 1 9 7 |
   | 6 9 2 | 1 8 7 | 4 5 3 |
   | 4 7 1 | 9 5 3 | 8 6 2 |
   ------------------------
   | 5 4 7 | 3 6 1 | 2 8 9 |
   | 2 8 3 | 5 9 4 | 6 7 1 |
   | 9 1 6 | 8 7 2 | 5 3 4 |
   ------------------------


You Won!!!


Woah! Your score is 255

ENTER ANY KEY TO GO BACK TO THE MAIN MENU......




\\----- GAME IS NOW EXITING -----//

C:\Users\Latitude E7440\source\repos\Project7\Debug\Project7.exe (process 12292) exited with code 0.
Press any key to close this window . . .
```

Option 1:

Test Case 2:

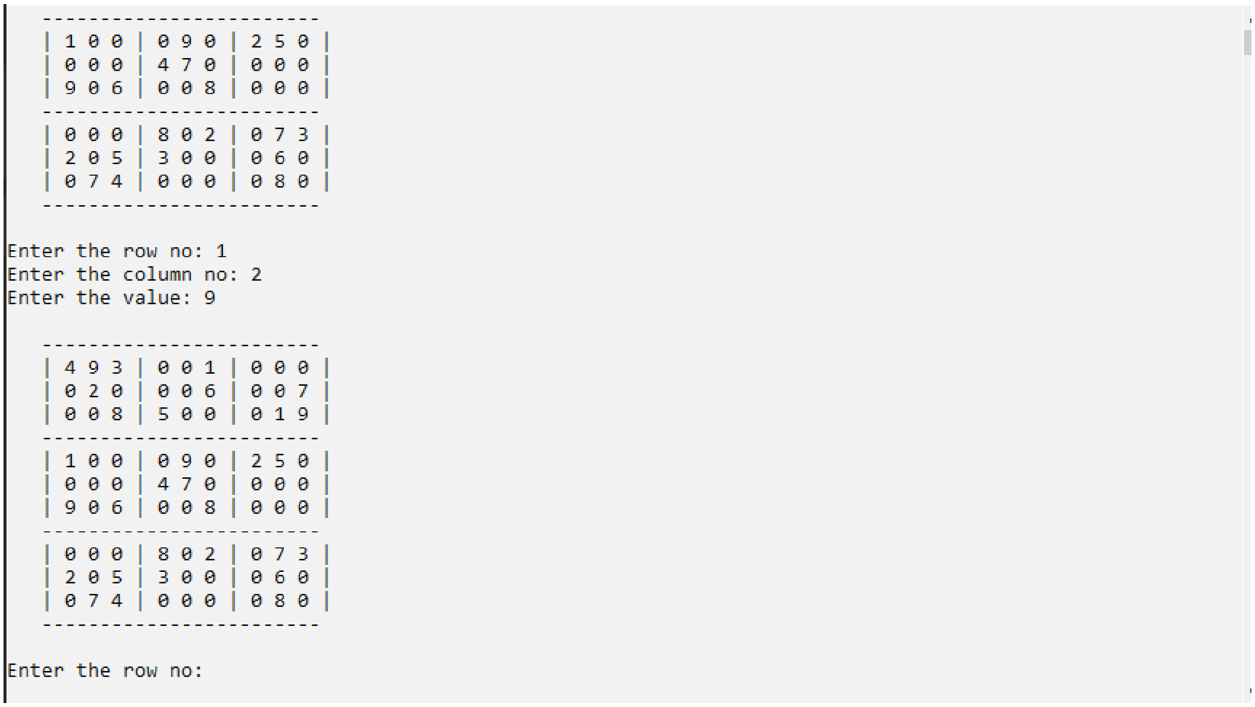
Outputs:

//Sudoku given to user to fill the missing(0’s)



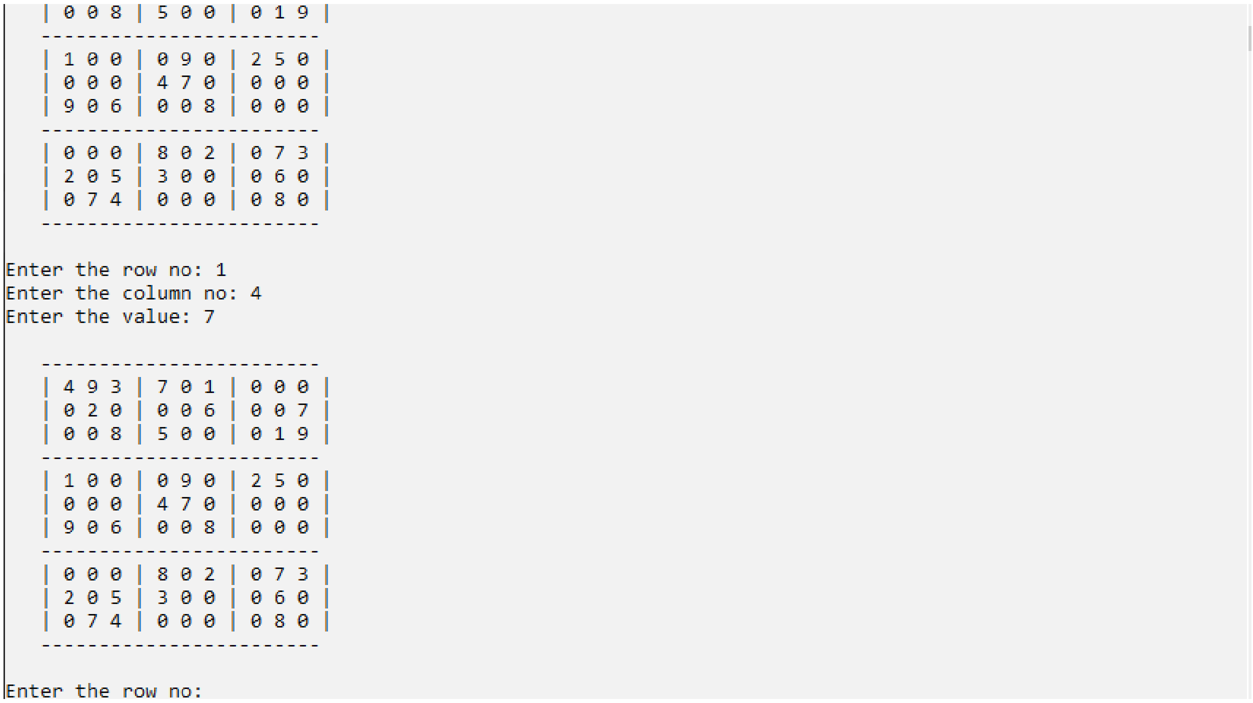
```
   ------------------------
   | 4 0 3 | 0 0 1 | 0 0 0 |
   | 0 2 0 | 0 0 6 | 0 0 7 |
   | 0 0 8 | 5 0 0 | 0 1 9 |
   ------------------------
   | 1 0 0 | 0 9 0 | 2 5 0 |
   | 0 0 0 | 4 7 0 | 0 0 0 |
   | 9 0 6 | 0 0 8 | 0 0 0 |
   ------------------------
   | 0 0 0 | 8 0 2 | 0 7 3 |
   | 2 0 5 | 3 0 0 | 0 6 0 |
   | 0 7 4 | 0 0 0 | 0 8 0 |
   ------------------------
```

//Asks for row,col and the value to be inputted there (1)



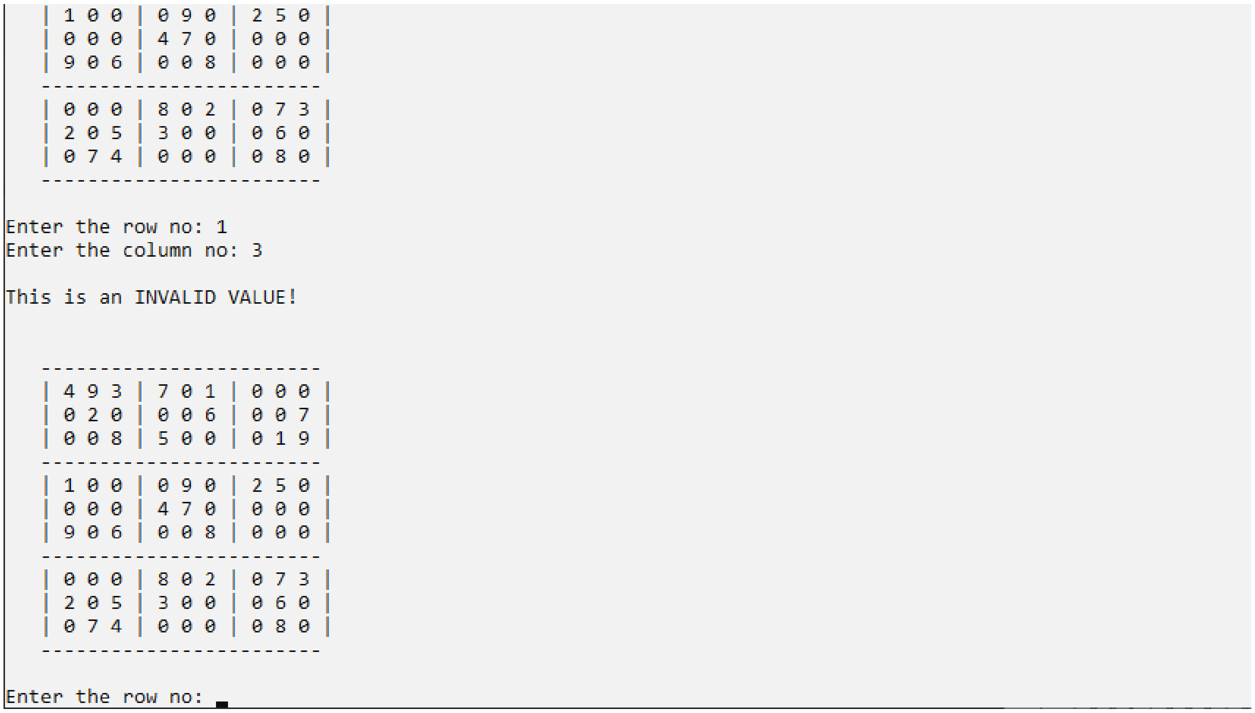
```
   -------------------------
   | 1 0 0 | 0 9 0 | 2 5 0 |
   | 0 0 0 | 4 7 0 | 0 0 0 |
   | 9 0 6 | 0 0 8 | 0 0 0 |
   -------------------------
   | 0 0 0 | 8 0 2 | 0 7 3 |
   | 2 0 5 | 3 0 0 | 0 6 0 |
   | 0 7 4 | 0 0 0 | 0 8 0 |
   -------------------------

Enter the row no: 1
Enter the column no: 2
Enter the value: 9

   -------------------------
   | 4 9 3 | 0 0 1 | 0 0 0 |
   | 0 2 0 | 0 0 6 | 0 0 7 |
   | 0 0 8 | 5 0 0 | 0 1 9 |
   -------------------------
   | 1 0 0 | 0 9 0 | 2 5 0 |
   | 0 0 0 | 4 7 0 | 0 0 0 |
   | 9 0 6 | 0 0 8 | 0 0 0 |
   -------------------------
   | 0 0 0 | 8 0 2 | 0 7 3 |
   | 2 0 5 | 3 0 0 | 0 6 0 |
   | 0 7 4 | 0 0 0 | 0 8 0 |
   -------------------------

Enter the row no:
```

//Asks for row,col and the value to be inputted there (2)



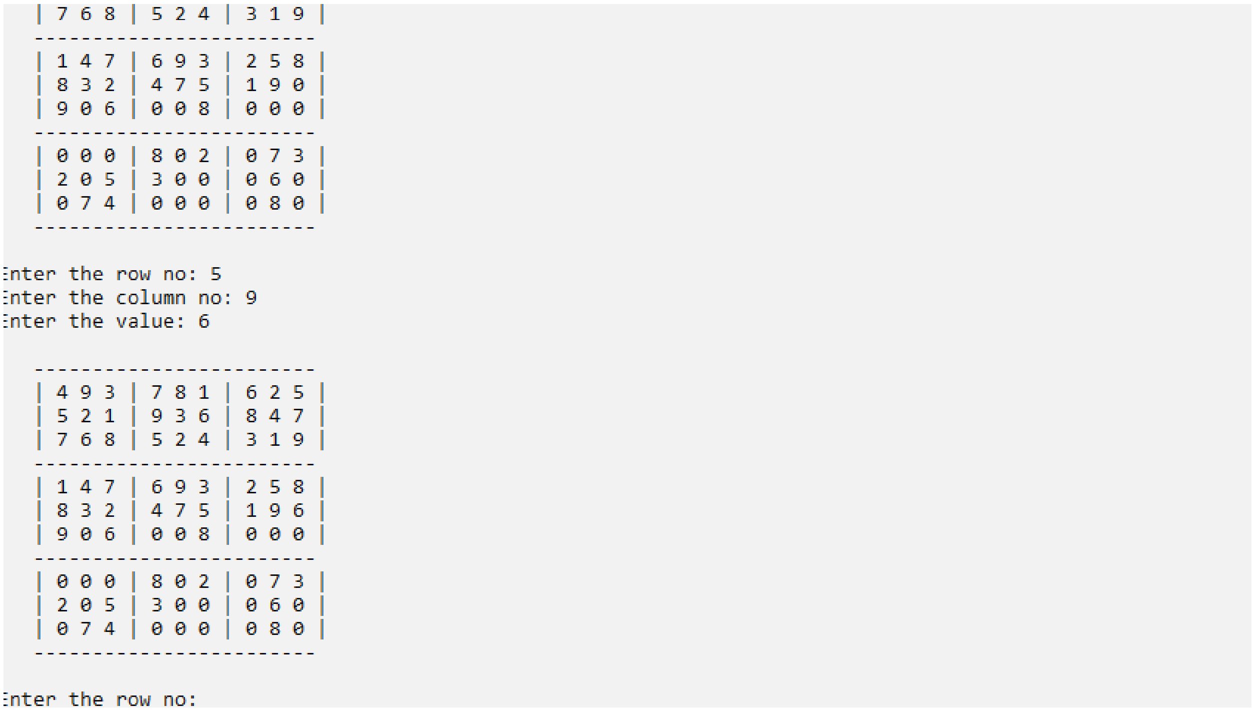
```
   | 0 0 8 | 5 0 0 | 0 1 9 |
   -------------------------
   | 1 0 0 | 0 9 0 | 2 5 0 |
   | 0 0 0 | 4 7 0 | 0 0 0 |
   | 9 0 6 | 0 0 8 | 0 0 0 |
   -------------------------
   | 0 0 0 | 8 0 2 | 0 7 3 |
   | 2 0 5 | 3 0 0 | 0 6 0 |
   | 0 7 4 | 0 0 0 | 0 8 0 |
   -------------------------

Enter the row no: 1
Enter the column no: 4
Enter the value: 7

   -------------------------
   | 4 9 3 | 7 0 1 | 0 0 0 |
   | 0 2 0 | 0 0 6 | 0 0 7 |
   | 0 0 8 | 5 0 0 | 0 1 9 |
   -------------------------
   | 1 0 0 | 0 9 0 | 2 5 0 |
   | 0 0 0 | 4 7 0 | 0 0 0 |
   | 9 0 6 | 0 0 8 | 0 0 0 |
   -------------------------
   | 0 0 0 | 8 0 2 | 0 7 3 |
   | 2 0 5 | 3 0 0 | 0 6 0 |
   | 0 7 4 | 0 0 0 | 0 8 0 |
   -------------------------

Enter the row no:
```

//For an Invalid Entry



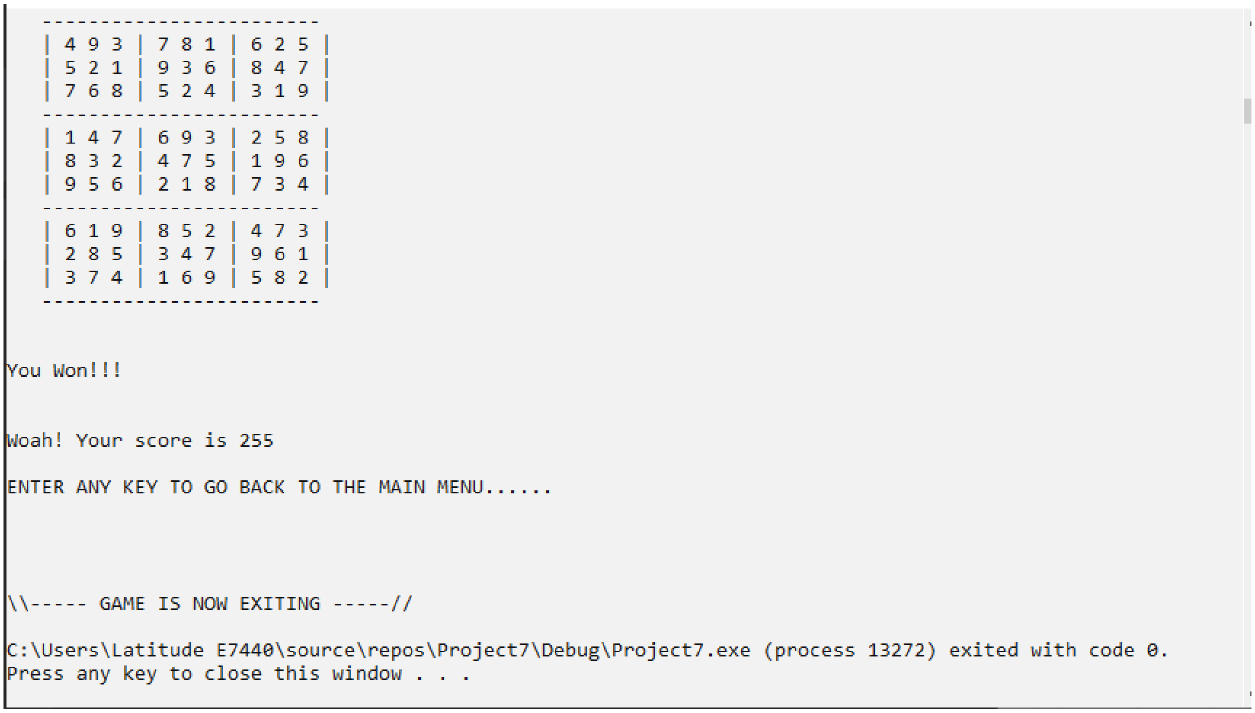
```
   | 1 0 0 | 0 9 0 | 2 5 0 |
   | 0 0 0 | 4 7 0 | 0 0 0 |
   | 9 0 6 | 0 0 8 | 0 0 0 |
   -------------------------
   | 0 0 0 | 8 0 2 | 0 7 3 |
   | 2 0 5 | 3 0 0 | 0 6 0 |
   | 0 7 4 | 0 0 0 | 0 8 0 |
   -------------------------

Enter the row no: 1
Enter the column no: 3

This is an INVALID VALUE!


   -------------------------
   | 4 9 3 | 7 0 1 | 0 0 0 |
   | 0 2 0 | 0 0 6 | 0 0 7 |
   | 0 0 8 | 5 0 0 | 0 1 9 |
   -------------------------
   | 1 0 0 | 0 9 0 | 2 5 0 |
   | 0 0 0 | 4 7 0 | 0 0 0 |
   | 9 0 6 | 0 0 8 | 0 0 0 |
   -------------------------
   | 0 0 0 | 8 0 2 | 0 7 3 |
   | 2 0 5 | 3 0 0 | 0 6 0 |
   | 0 7 4 | 0 0 0 | 0 8 0 |
   -------------------------

Enter the row no: _
```

//Intermediate Result



```
   | 7 6 8 | 5 2 4 | 3 1 9 |
   -------------------------
   | 1 4 7 | 6 9 3 | 2 5 8 |
   | 8 3 2 | 4 7 5 | 1 9 0 |
   | 9 0 6 | 0 0 8 | 0 0 0 |
   -------------------------
   | 0 0 0 | 8 0 2 | 0 7 3 |
   | 2 0 5 | 3 0 0 | 0 6 0 |
   | 0 7 4 | 0 0 0 | 0 8 0 |
   -------------------------

Enter the row no: 5
Enter the column no: 9
Enter the value: 6

   -------------------------
   | 4 9 3 | 7 8 1 | 6 2 5 |
   | 5 2 1 | 9 3 6 | 8 4 7 |
   | 7 6 8 | 5 2 4 | 3 1 9 |
   -------------------------
   | 1 4 7 | 6 9 3 | 2 5 8 |
   | 8 3 2 | 4 7 5 | 1 9 6 |
   | 9 0 6 | 0 0 8 | 0 0 0 |
   -------------------------
   | 0 0 0 | 8 0 2 | 0 7 3 |
   | 2 0 5 | 3 0 0 | 0 6 0 |
   | 0 7 4 | 0 0 0 | 0 8 0 |
   -------------------------

Enter the row no:
```

//Final Result

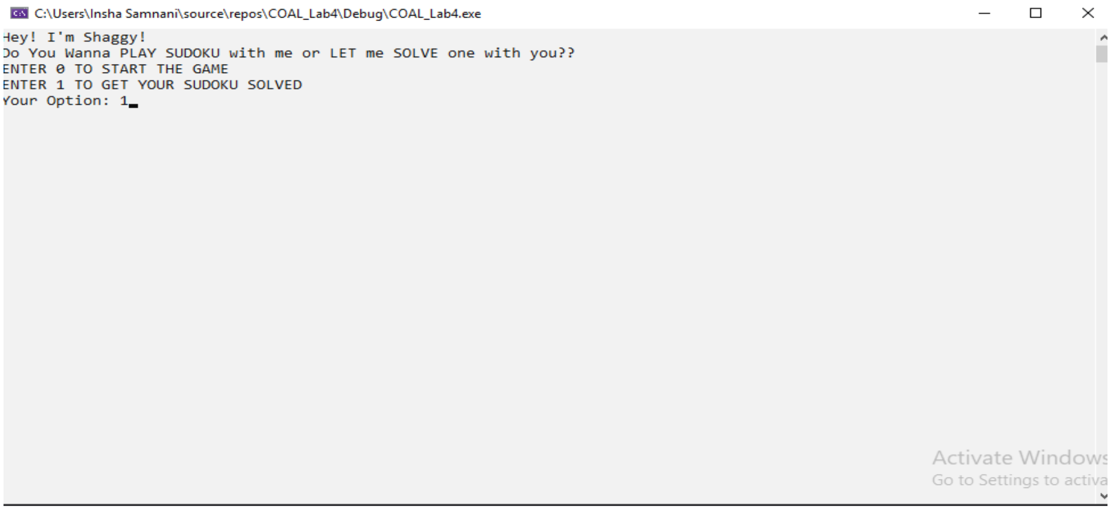


```
   -----------------------
   | 4 9 3 | 7 8 1 | 6 2 5 |
   | 5 2 1 | 9 3 6 | 8 4 7 |
   | 7 6 8 | 5 2 4 | 3 1 9 |
   -----------------------
   | 1 4 7 | 6 9 3 | 2 5 8 |
   | 8 3 2 | 4 7 5 | 1 9 6 |
   | 9 5 6 | 2 1 8 | 7 3 4 |
   -----------------------
   | 6 1 9 | 8 5 2 | 4 7 3 |
   | 2 8 5 | 3 4 7 | 9 6 1 |
   | 3 7 4 | 1 6 9 | 5 8 2 |
   -----------------------


You Won!!!


Woah! Your score is 255

ENTER ANY KEY TO GO BACK TO THE MAIN MENU......




\\----- GAME IS NOW EXITING -----//

C:\Users\Latitude E7440\source\repos\Project7\Debug\Project7.exe (process 13272) exited with code 0.
Press any key to close this window . . .
```

Option 2:

Test Case 1:

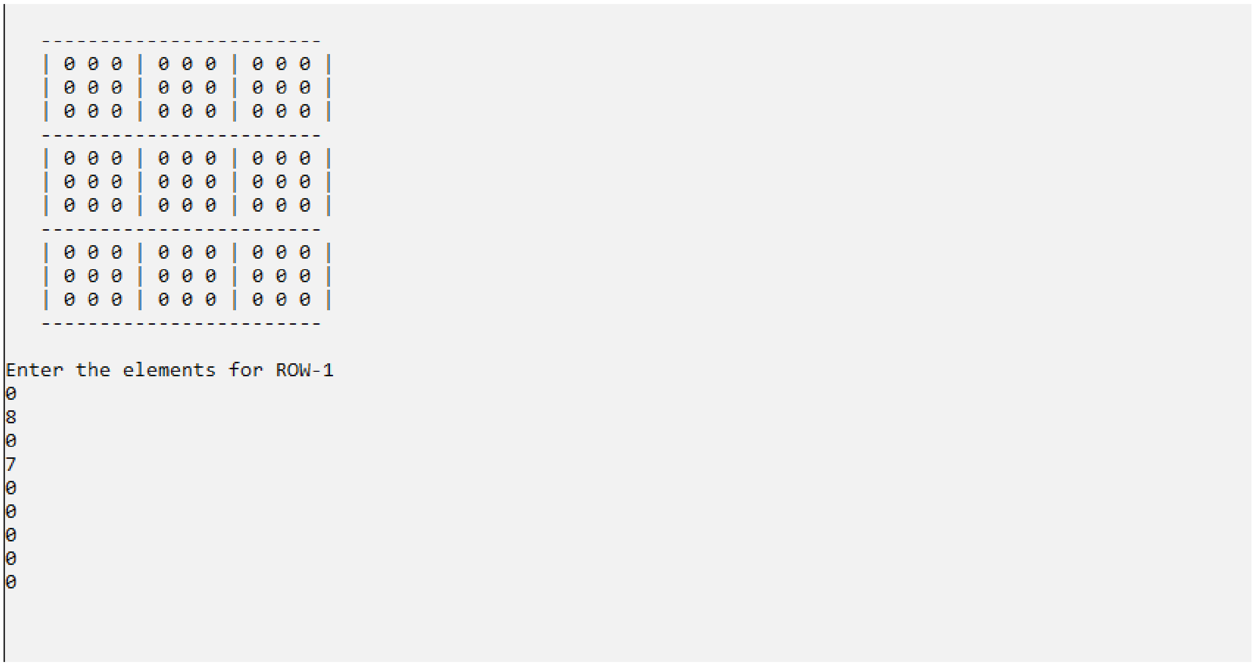
Outputs:


C:\Users\Insha Samnani\source\repos\COAL_Lab4\Debug\COAL_Lab4.exe
Activate Windows
Go to Settings to activa

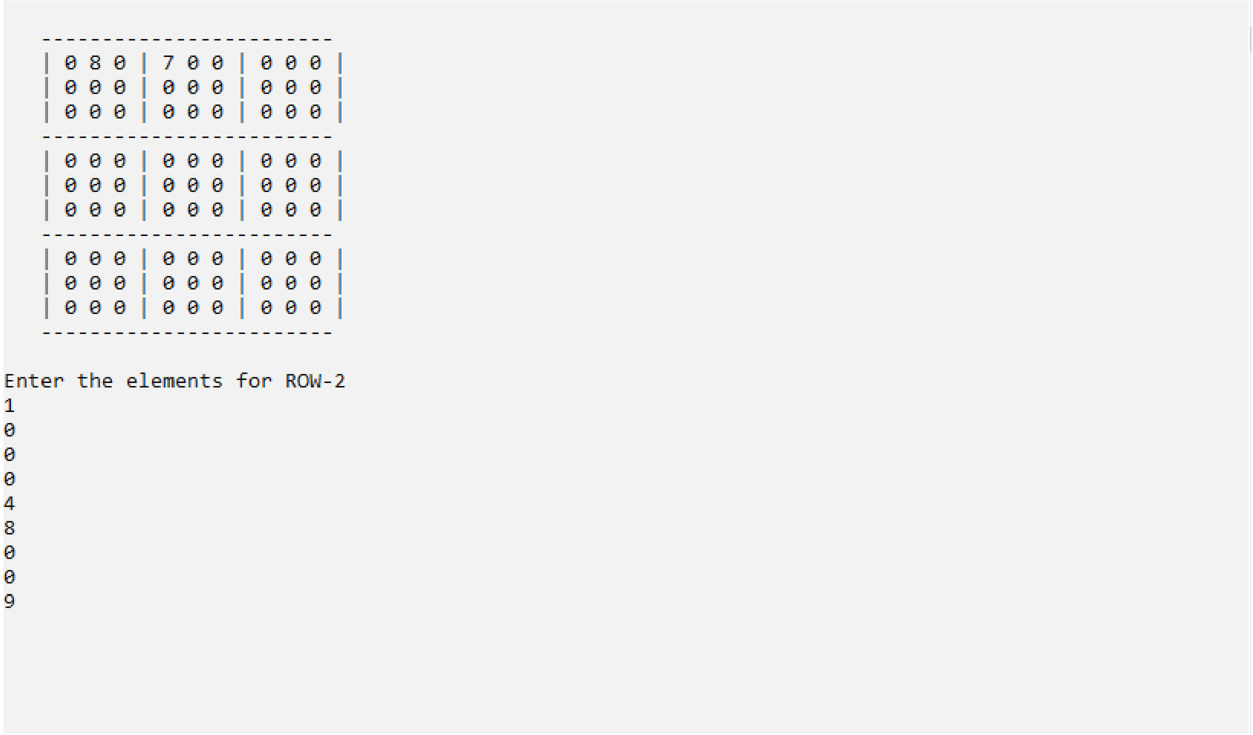
```
Hey! I'm Shaggy!
Do You Wanna PLAY SUDOKU with me or LET me SOLVE one with you??
ENTER 0 TO START THE GAME
ENTER 1 TO GET YOUR SUDOKU SOLVED
Your Option: 1
```

//A 9*9 grid of all o’s(empty) grid given to fill

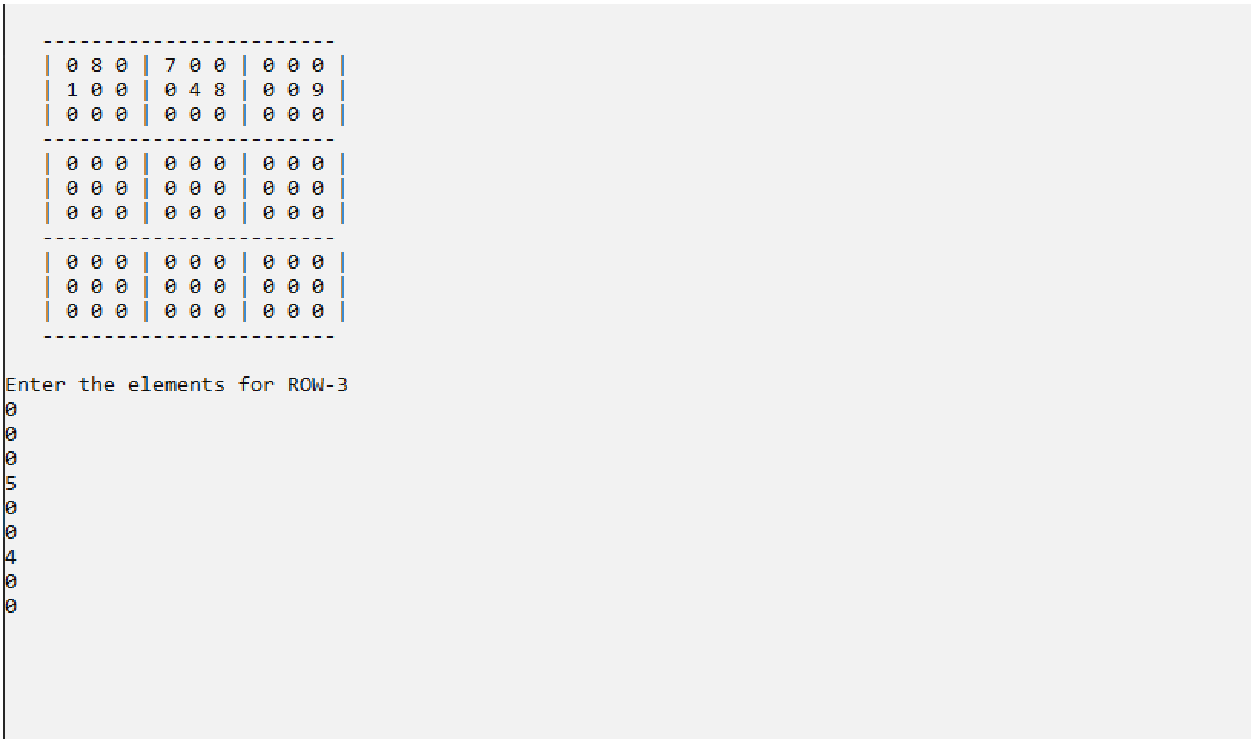
//Row by row entering of values



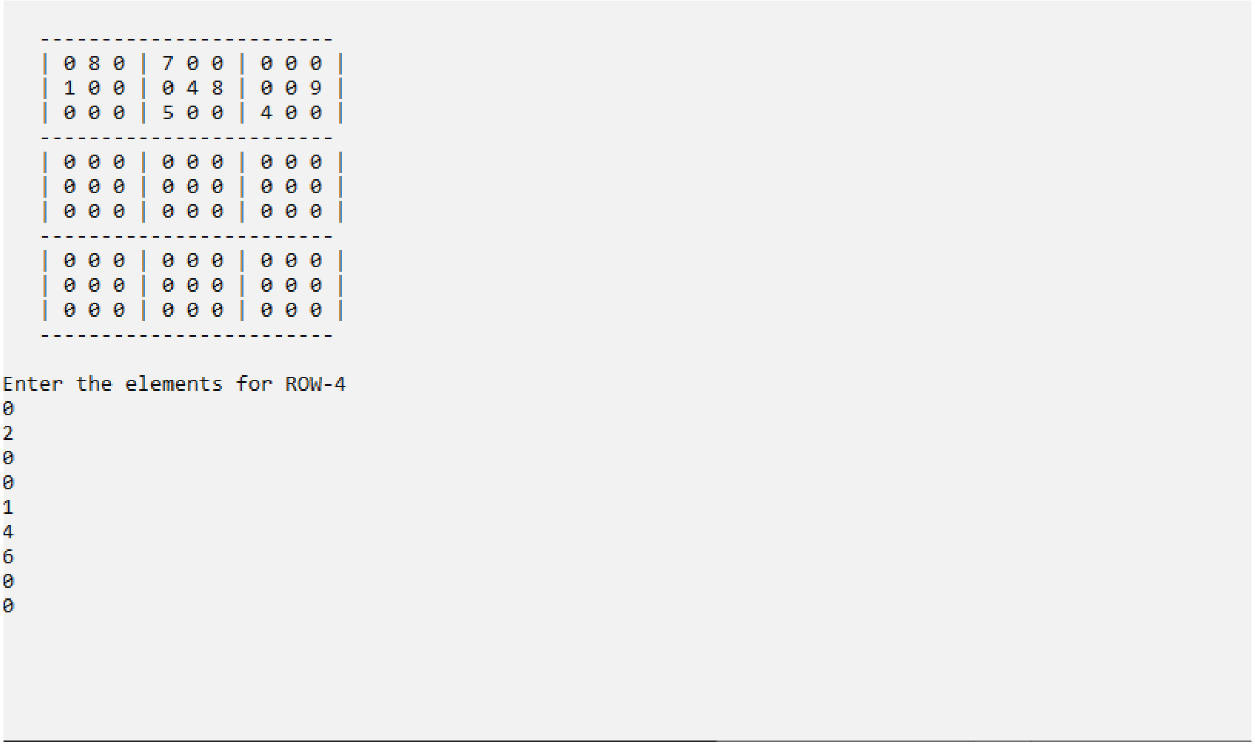
```
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------

Enter the elements for ROW-1
0
8
0
7
0
0
0
0
0
```



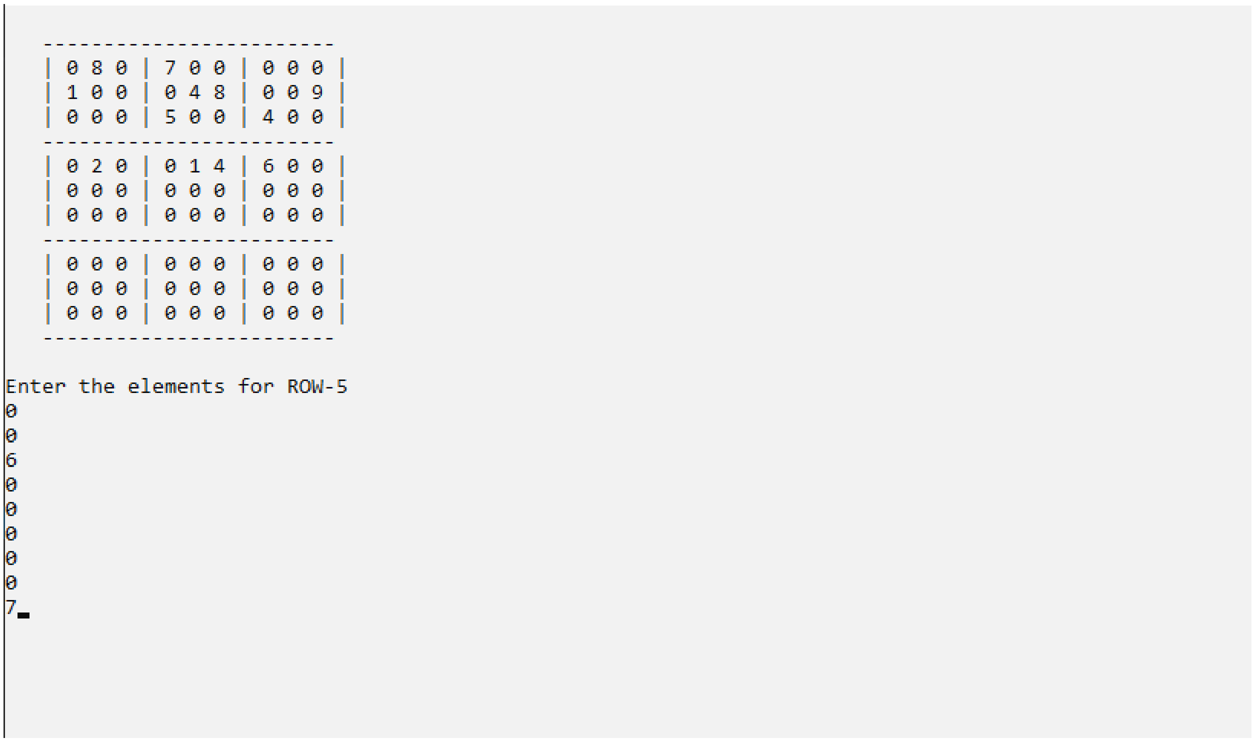
```
   -------------------------
   | 0 8 0 | 7 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------

Enter the elements for ROW-2
1
0
0
0
4
8
0
0
9
```



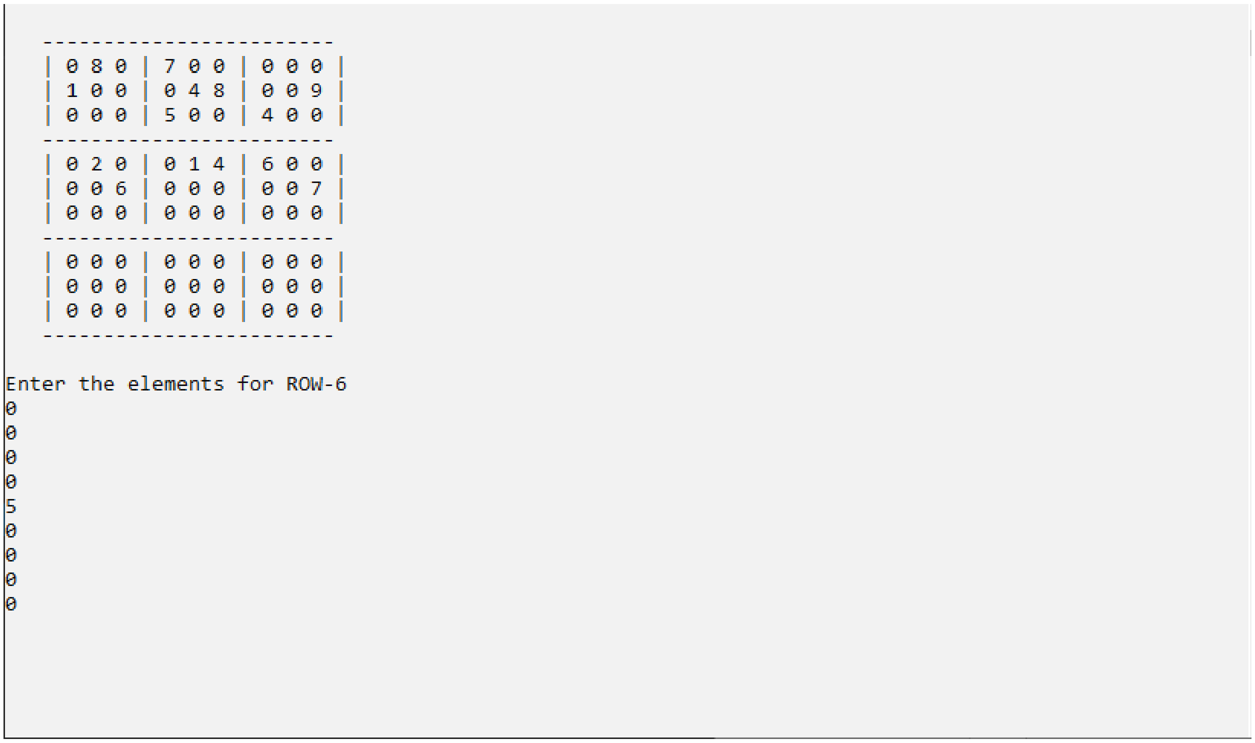
```
   -------------------------
   | 0 8 0 | 7 0 0 | 0 0 0 |
   | 1 0 0 | 0 4 8 | 0 0 9 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------

Enter the elements for ROW-3
0
0
0
5
0
0
4
0
0
```



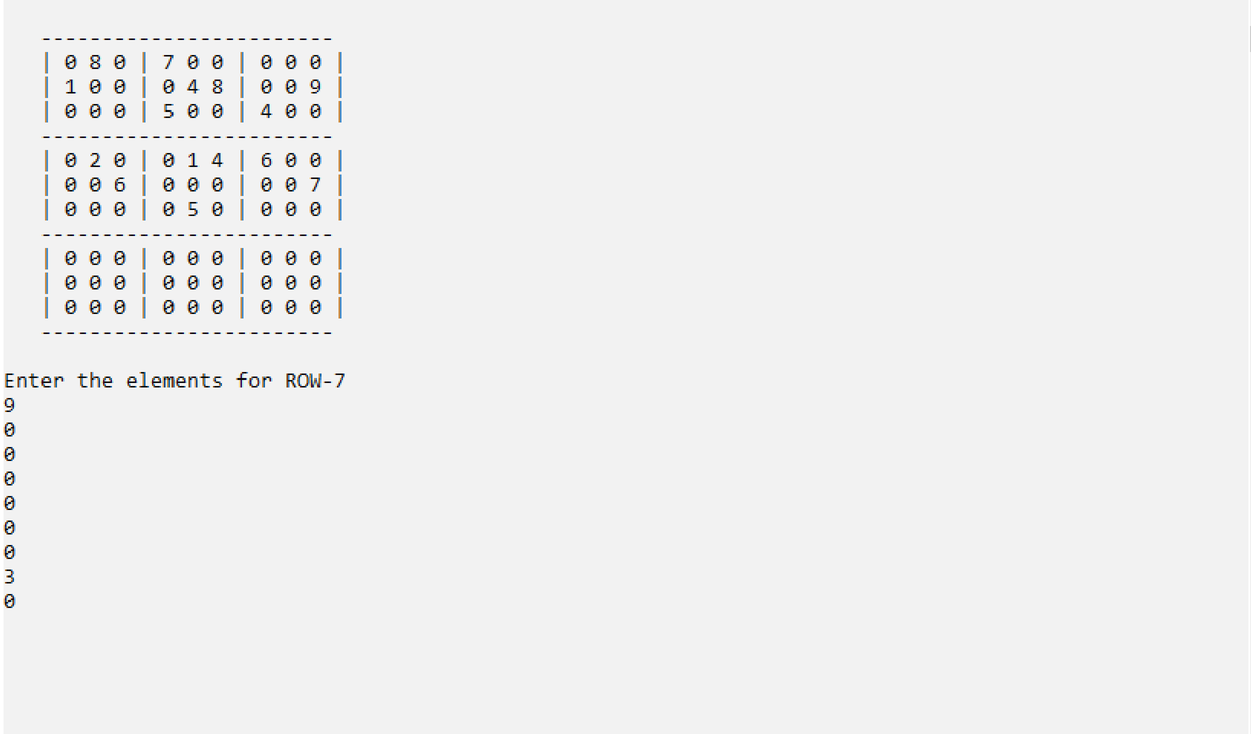
```
   -------------------------
   | 0 8 0 | 7 0 0 | 0 0 0 |
   | 1 0 0 | 0 4 8 | 0 0 9 |
   | 0 0 0 | 5 0 0 | 4 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------

Enter the elements for ROW-4
0
2
0
0
1
4
6
0
0
```



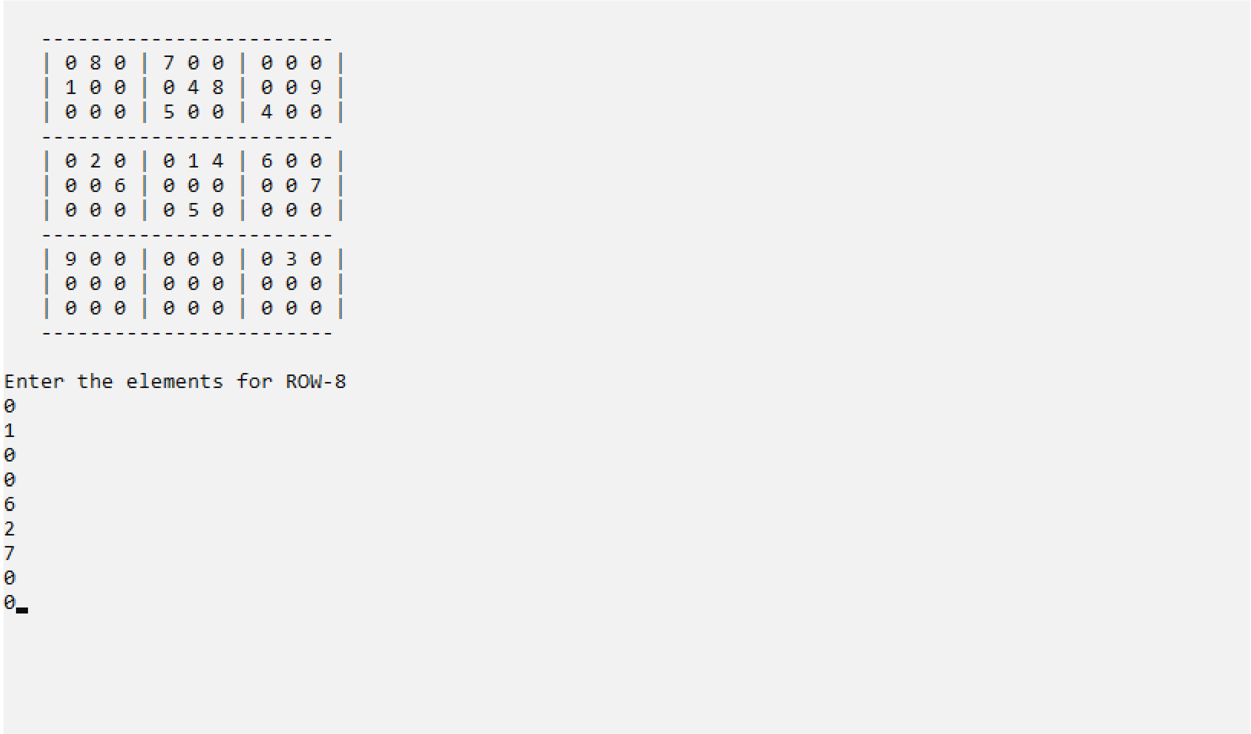
```
   -------------------------
   | 0 8 0 | 7 0 0 | 0 0 0 |
   | 1 0 0 | 0 4 8 | 0 0 9 |
   | 0 0 0 | 5 0 0 | 4 0 0 |
   -------------------------
   | 0 2 0 | 0 1 4 | 6 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------

Enter the elements for ROW-5
0
0
6
0
0
0
0
0
7
```



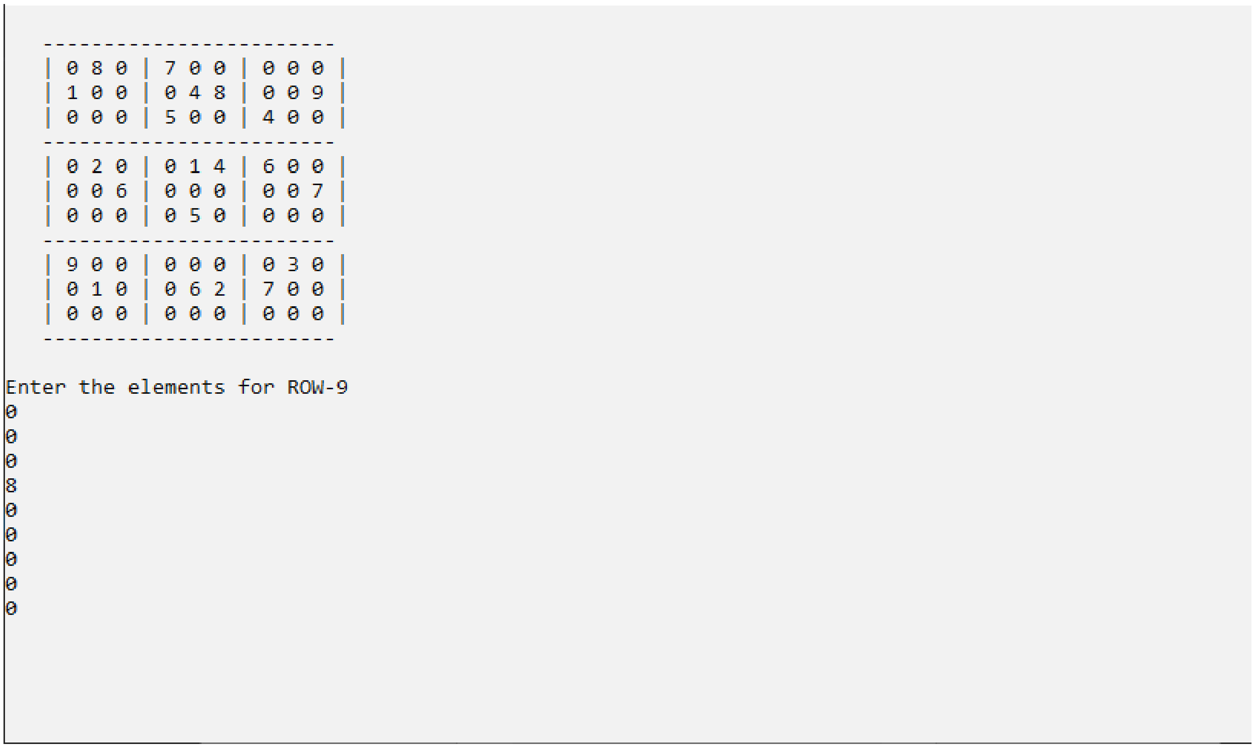
```
   -------------------------
   | 0 8 0 | 7 0 0 | 0 0 0 |
   | 1 0 0 | 0 4 8 | 0 0 9 |
   | 0 0 0 | 5 0 0 | 4 0 0 |
   -------------------------
   | 0 2 0 | 0 1 4 | 6 0 0 |
   | 0 0 6 | 0 0 0 | 0 0 7 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------

Enter the elements for ROW-6
0
0
0
0
5
0
0
0
0
```



```
   ------------------------
   | 0 8 0 | 7 0 0 | 0 0 0 |
   | 1 0 0 | 0 4 8 | 0 0 9 |
   | 0 0 0 | 5 0 0 | 4 0 0 |
   ------------------------
   | 0 2 0 | 0 1 4 | 6 0 0 |
   | 0 0 6 | 0 0 0 | 0 0 7 |
   | 0 0 0 | 0 5 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------

Enter the elements for ROW-7
9
0
0
0
0
0
0
3
0
```

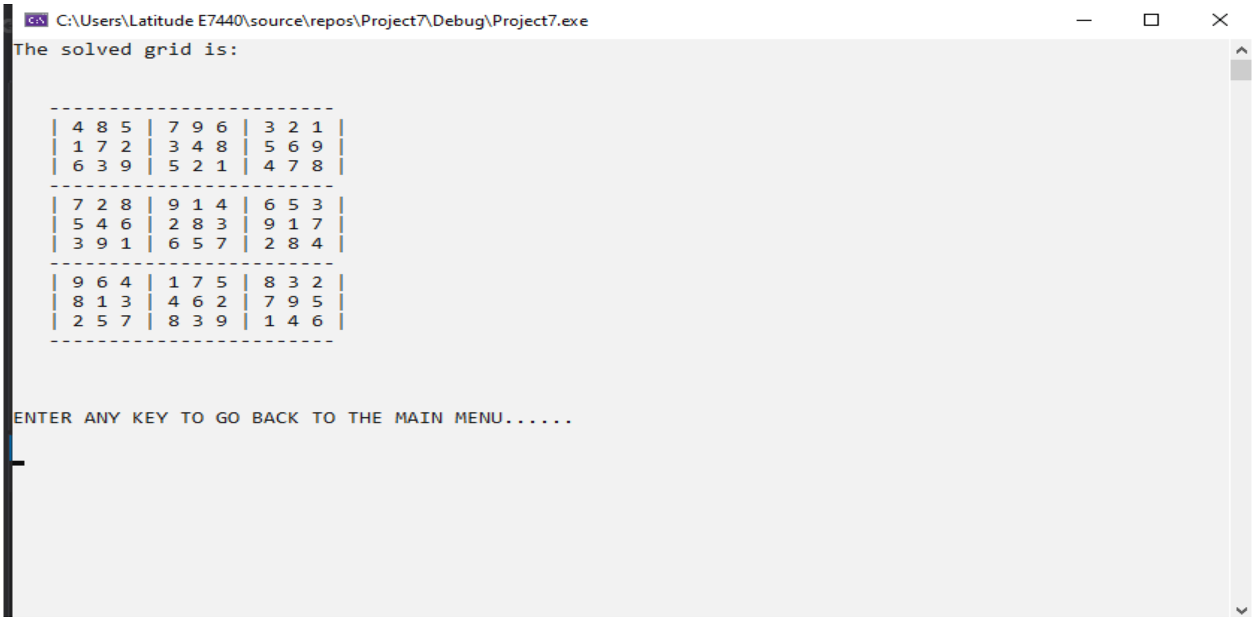


```
   ------------------------
   | 0 8 0 | 7 0 0 | 0 0 0 |
   | 1 0 0 | 0 4 8 | 0 0 9 |
   | 0 0 0 | 5 0 0 | 4 0 0 |
   ------------------------
   | 0 2 0 | 0 1 4 | 6 0 0 |
   | 0 0 6 | 0 0 0 | 0 0 7 |
   | 0 0 0 | 0 5 0 | 0 0 0 |
   ------------------------
   | 9 0 0 | 0 0 0 | 0 3 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------

Enter the elements for ROW-8
0
1
0
0
6
2
7
0
0
```



```
   -------------------------
   | 0 8 0 | 7 0 0 | 0 0 0 |
   | 1 0 0 | 0 4 8 | 0 0 9 |
   | 0 0 0 | 5 0 0 | 4 0 0 |
   -------------------------
   | 0 2 0 | 0 1 4 | 6 0 0 |
   | 0 0 6 | 0 0 0 | 0 0 7 |
   | 0 0 0 | 0 5 0 | 0 0 0 |
   -------------------------
   | 9 0 0 | 0 0 0 | 0 3 0 |
   | 0 1 0 | 0 6 2 | 7 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------

Enter the elements for ROW-9
0
0
0
8
0
0
0
0
0
```

//Sudoku Solved by the Program



C:\Users\Latitude E7440\source\repos\Project7\Debug\Project7.exe

```
The solved grid is:


   -------------------------
   | 4 8 5 | 7 9 6 | 3 2 1 |
   | 1 7 2 | 3 4 8 | 5 6 9 |
   | 6 3 9 | 5 2 1 | 4 7 8 |
   -------------------------
   | 7 2 8 | 9 1 4 | 6 5 3 |
   | 5 4 6 | 2 8 3 | 9 1 7 |
   | 3 9 1 | 6 5 7 | 2 8 4 |
   -------------------------
   | 9 6 4 | 1 7 5 | 8 3 2 |
   | 8 1 3 | 4 6 2 | 7 9 5 |
   | 2 5 7 | 8 3 9 | 1 4 6 |
   -------------------------



ENTER ANY KEY TO GO BACK TO THE MAIN MENU......
```

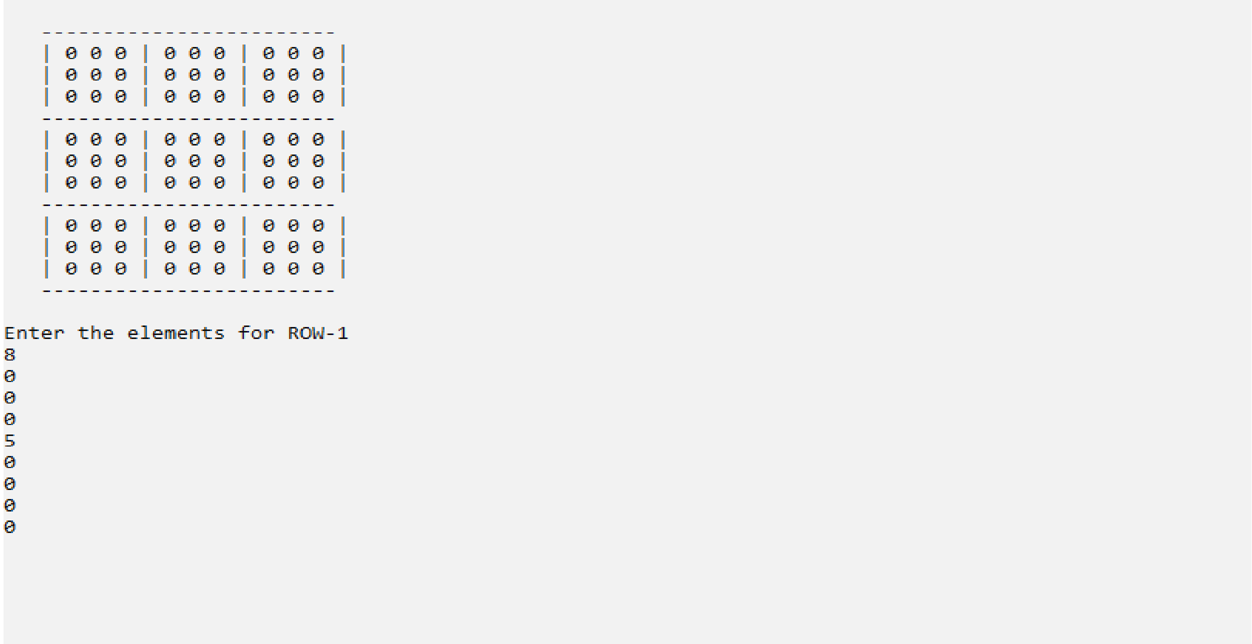
Option 2:

Test Case 2:

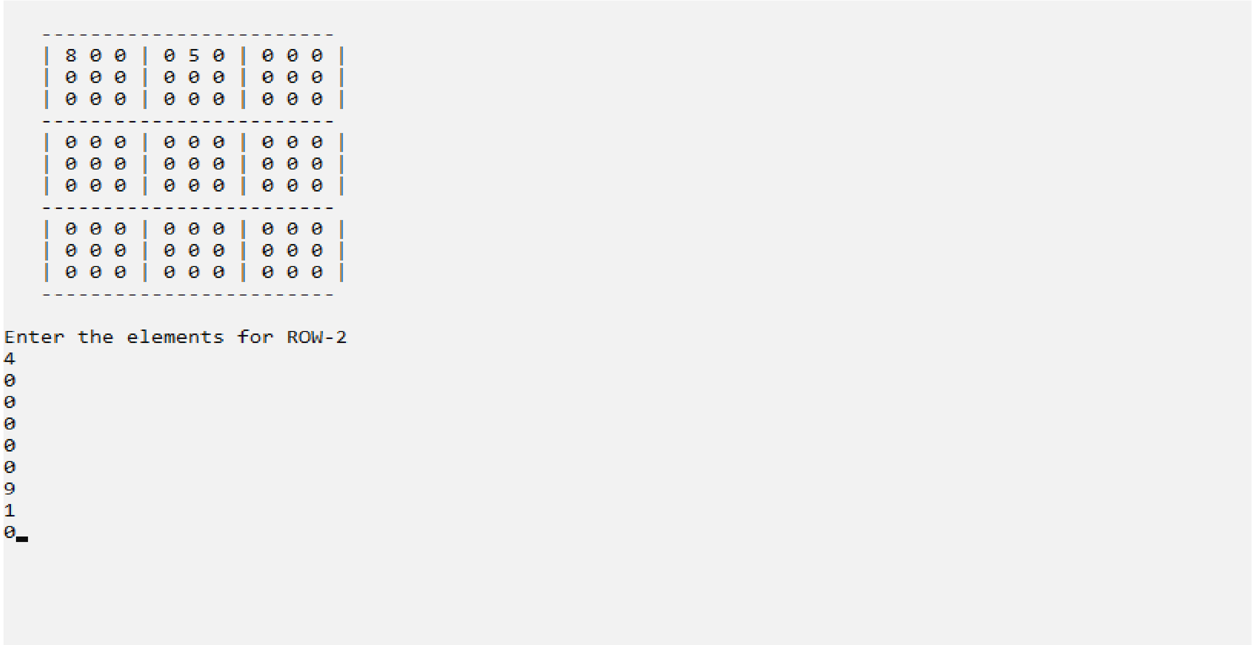
Outputs:

//A 9*9 grid of all o’s(empty) grid given to fill

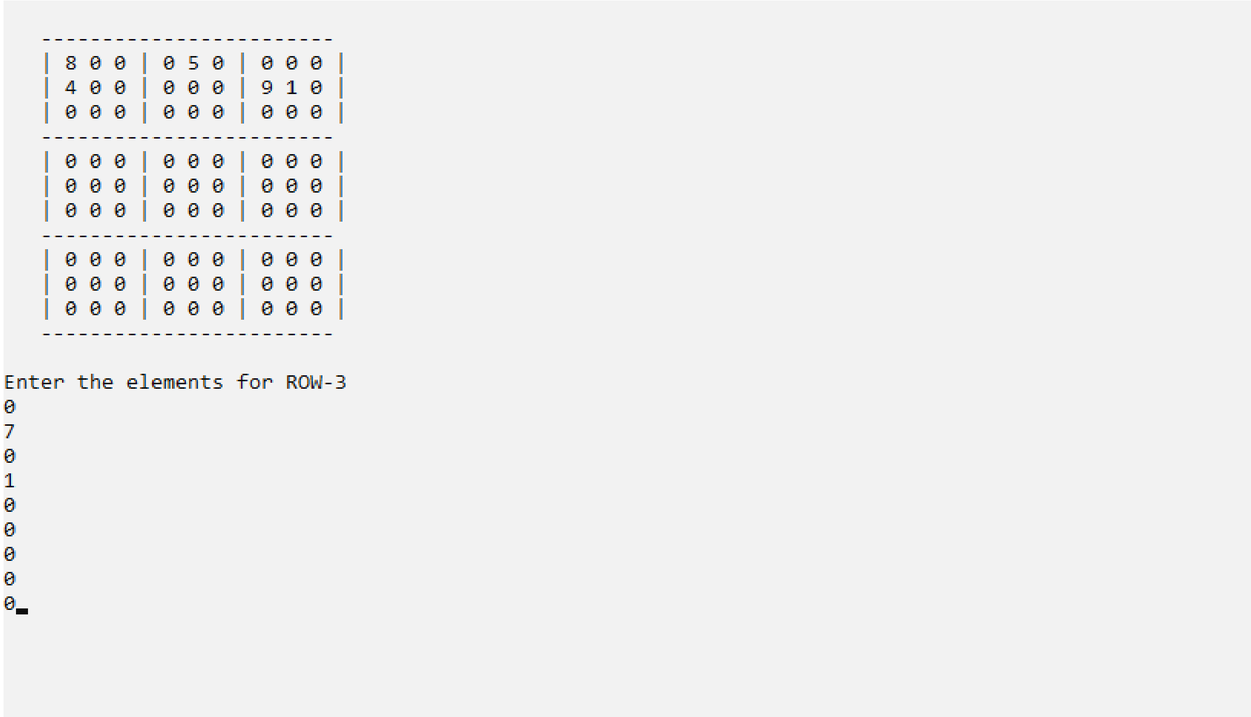
//Row by row entering of values



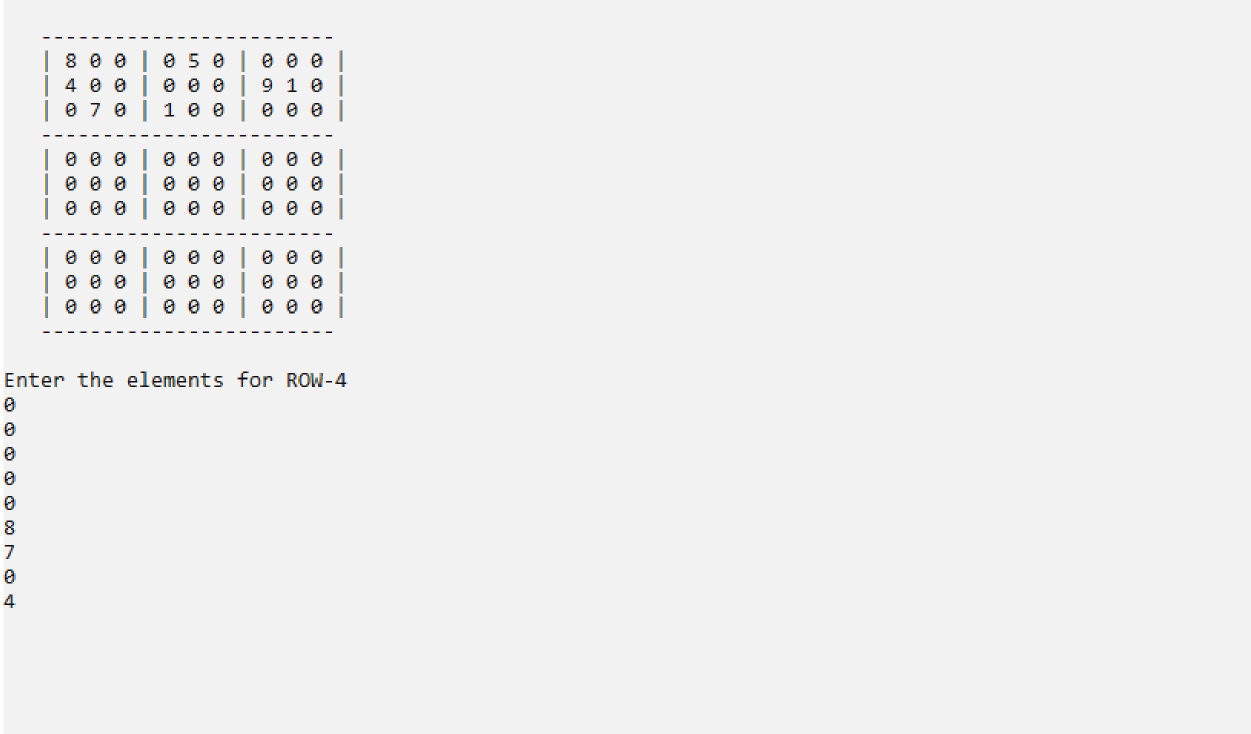
```
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------

Enter the elements for ROW-1
8
0
0
0
5
0
0
0
0
```



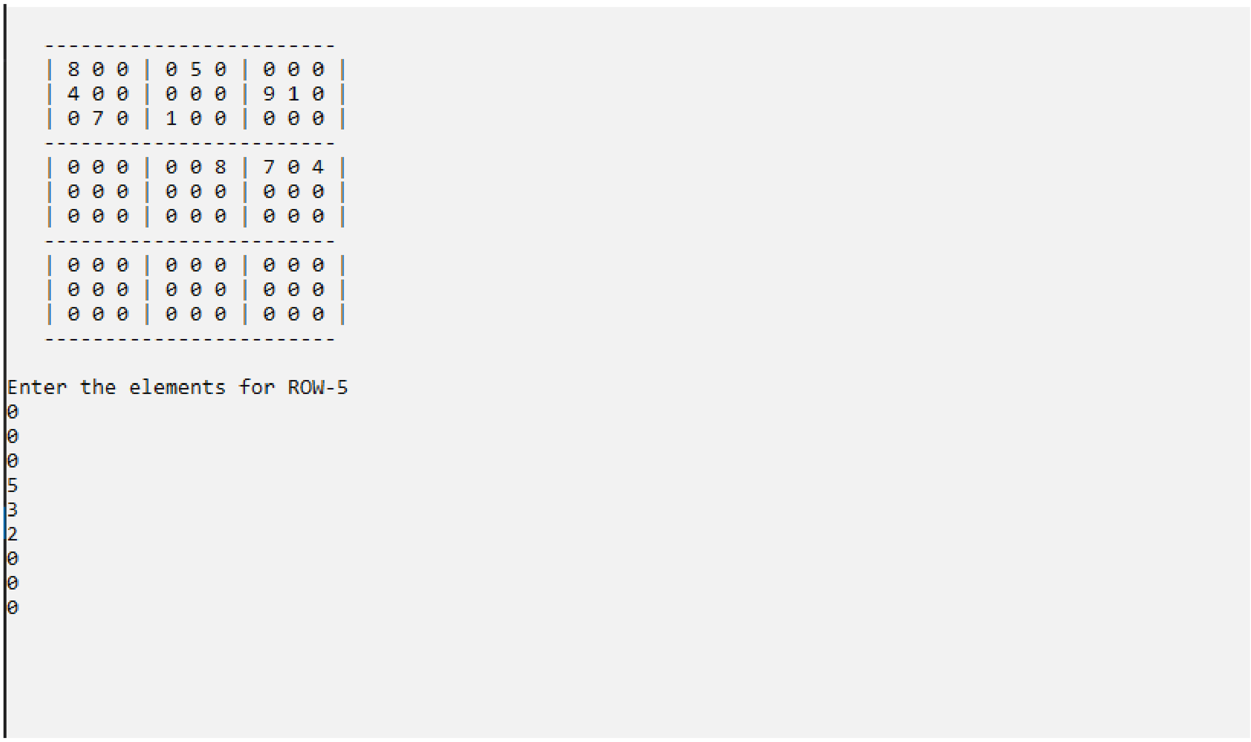
```
   ------------------------
   | 8 0 0 | 0 5 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------

Enter the elements for ROW-2
4
0
0
0
0
0
9
1
0
```



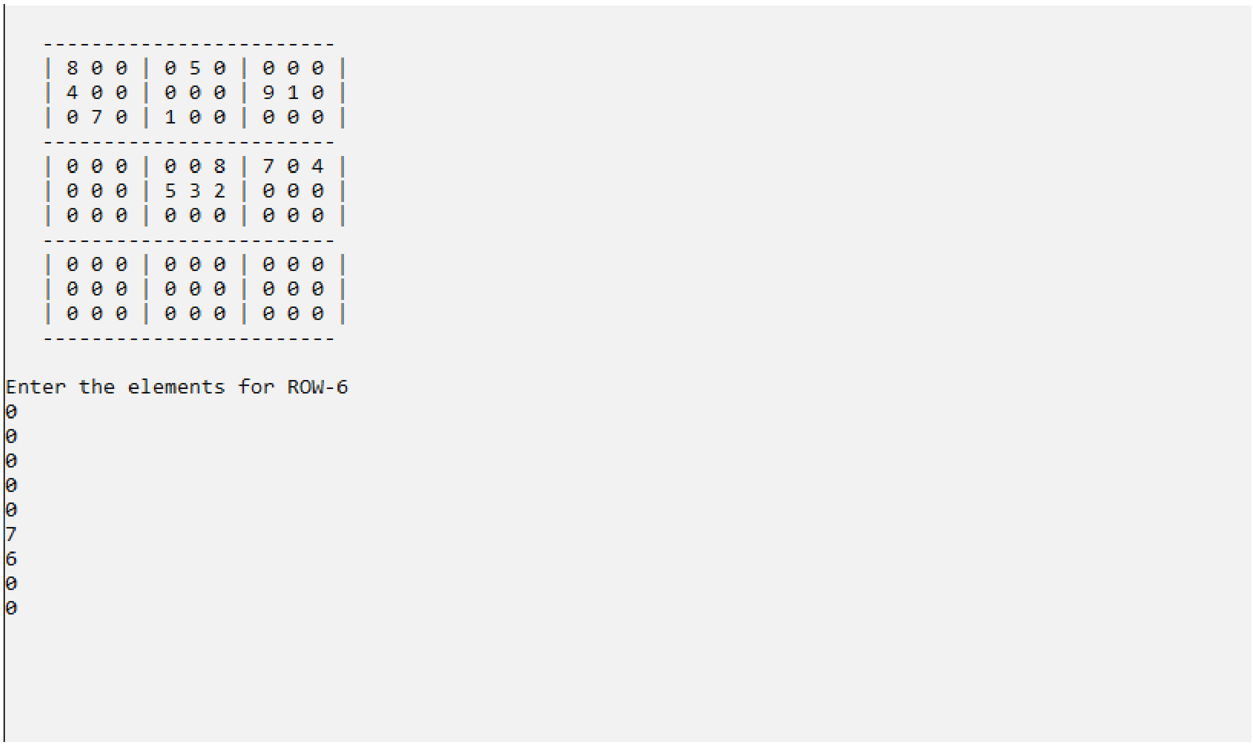
```
   ------------------------
   | 8 0 0 | 0 5 0 | 0 0 0 |
   | 4 0 0 | 0 0 0 | 9 1 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------

Enter the elements for ROW-3
0
7
0
1
0
0
0
0
0
```



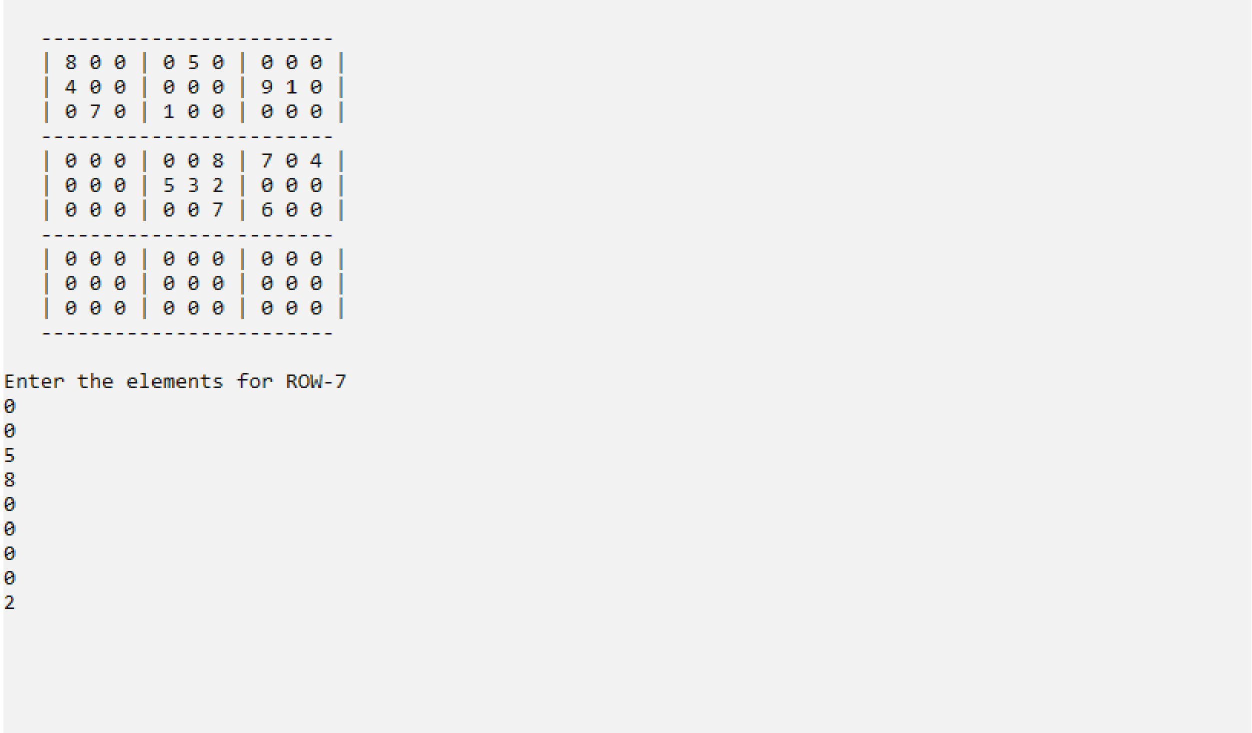
```
   ------------------------
   | 8 0 0 | 0 5 0 | 0 0 0 |
   | 4 0 0 | 0 0 0 | 9 1 0 |
   | 0 7 0 | 1 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------

Enter the elements for ROW-4
0
0
0
0
0
8
7
0
4
```



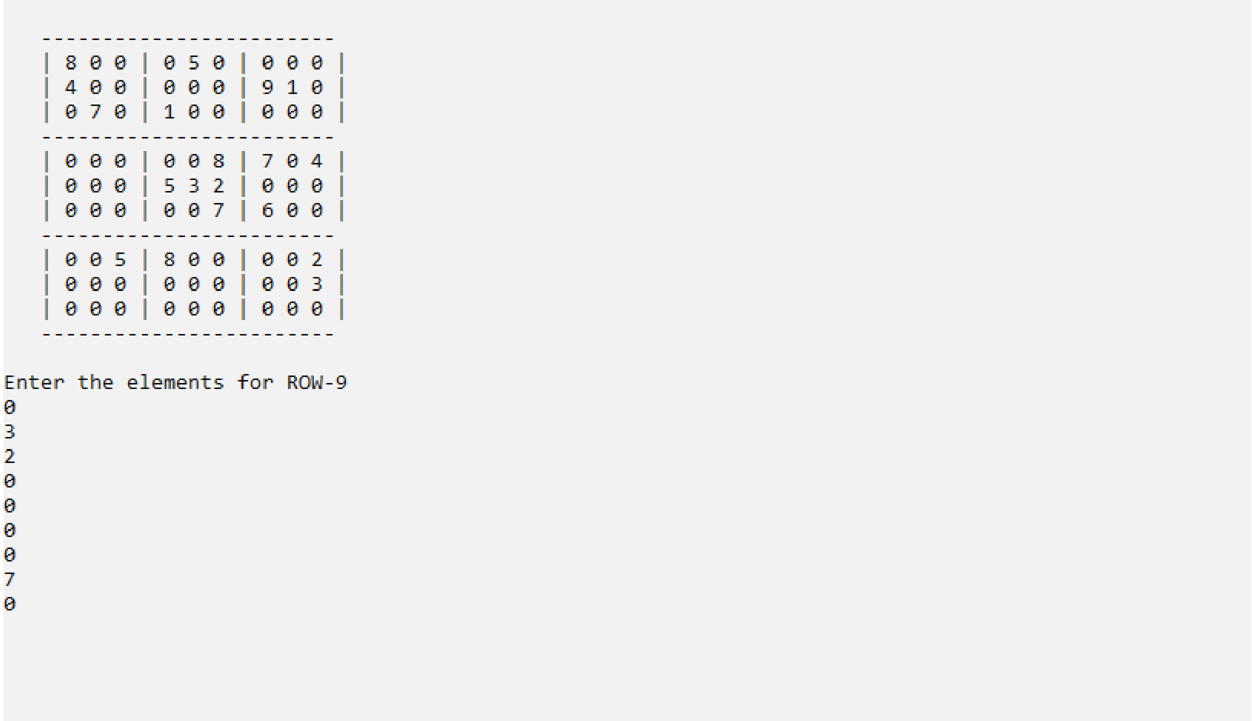
```
   ------------------------
   | 8 0 0 | 0 5 0 | 0 0 0 |
   | 4 0 0 | 0 0 0 | 9 1 0 |
   | 0 7 0 | 1 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 8 | 7 0 4 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------

Enter the elements for ROW-5
0
0
0
5
3
2
0
0
0
```



```
   ------------------------
   | 8 0 0 | 0 5 0 | 0 0 0 |
   | 4 0 0 | 0 0 0 | 9 1 0 |
   | 0 7 0 | 1 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 8 | 7 0 4 |
   | 0 0 0 | 5 3 2 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   ------------------------

Enter the elements for ROW-6
0
0
0
0
0
7
6
0
0
```

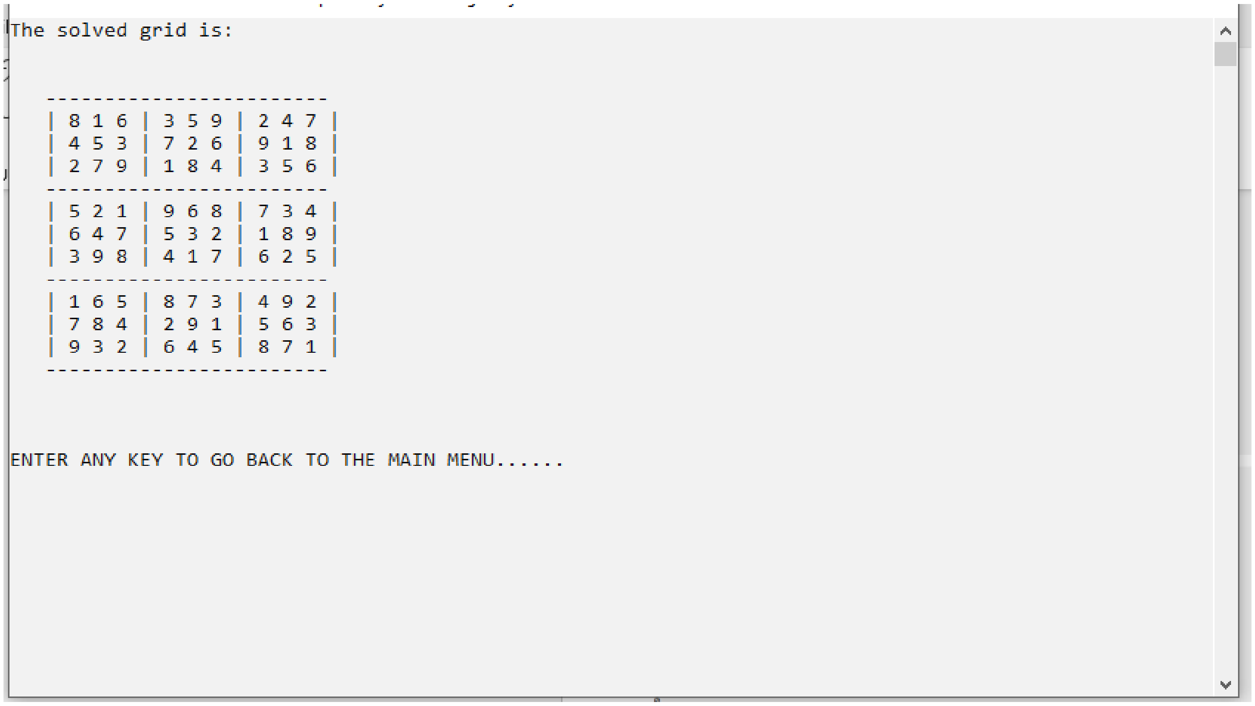


```
   -------------------------
   | 8 0 0 | 0 5 0 | 0 0 0 |
   | 4 0 0 | 0 0 0 | 9 1 0 |
   | 0 7 0 | 1 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 8 | 7 0 4 |
   | 0 0 0 | 5 3 2 | 0 0 0 |
   | 0 0 0 | 0 0 7 | 6 0 0 |
   -------------------------
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------

Enter the elements for ROW-7
0
0
5
8
0
0
0
0
2
```



```
   -------------------------
   | 8 0 0 | 0 5 0 | 0 0 0 |
   | 4 0 0 | 0 0 0 | 9 1 0 |
   | 0 7 0 | 1 0 0 | 0 0 0 |
   -------------------------
   | 0 0 0 | 0 0 8 | 7 0 4 |
   | 0 0 0 | 5 3 2 | 0 0 0 |
   | 0 0 0 | 0 0 7 | 6 0 0 |
   -------------------------
   | 0 0 5 | 8 0 0 | 0 0 2 |
   | 0 0 0 | 0 0 0 | 0 0 3 |
   | 0 0 0 | 0 0 0 | 0 0 0 |
   -------------------------

Enter the elements for ROW-9
0
3
2
0
0
0
0
7
0
```

//Sudoku Solved by the Program



```
The solved grid is:


   ------------------------
   | 8 1 6 | 3 5 9 | 2 4 7 |
   | 4 5 3 | 7 2 6 | 9 1 8 |
   | 2 7 9 | 1 8 4 | 3 5 6 |
   ------------------------
   | 5 2 1 | 9 6 8 | 7 3 4 |
   | 6 4 7 | 5 3 2 | 1 8 9 |
   | 3 9 8 | 4 1 7 | 6 2 5 |
   ------------------------
   | 1 6 5 | 8 7 3 | 4 9 2 |
   | 7 8 4 | 2 9 1 | 5 6 3 |
   | 9 3 2 | 6 4 5 | 8 7 1 |
   ------------------------



ENTER ANY KEY TO GO BACK TO THE MAIN MENU......
```

CONCLUSION, COST AND FUTURE WORK:

In brief, Soduko Mystifier accepts the challenge of solving easy to evil level Sudoku Puzzles by computing runtime values for the empty boxes. Not only it solves the puzzle, but it also does helps user to play Sudoku game and display its score at the end of solving the mystifier. Concentration, Memory, Learning, Relaxation are the attributes which are evaluated while solving a Sudoku. The conclusion to this Sudoku Mystifier is that it is optimized algorithm producing accurate results. Future advancements can be taken in advancement by expanding it to different matrix sizes i.e. (12*12,14*14, …,25*25).

REFERENCES:

1. https://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand13/Group1Vahid/report/Aref-Fiorella-KexJobb-sist.pdf
2. https://www.sudokuonline.io/tips/benefits-of-sudoku
3. https://undergroundmathematics.org/thinking-about-algebra/equation-sudoku/solution
4. https://www.geeksforgeeks.org/sudoku-backtracking-7/
5. https://towardsdatascience.com/solving-sudoku-with-ai-d6008993c7de
6. https://github.com/vindmi/asm-sudoku-solver/blob/master/sudoku.s