COAL Lab Project Report

National University of Computer and Emerging Sciences

Matrix Operations

Batch: 2023

Section: CY-3A

Group Members

Talal Ali (23K-2003)

Daniyal Ahmed (23K-2011)

Muhammad Hammad (23K-2005)

Lab Instructor

Mr. Waseem Rauf

Department of Cyber Security

National University of Computer and Emerging Sciences

– FAST Karachi Campus

Acknowledgement

We extend our heartfelt gratitude to Mr. Waseem Rauf for their guidance and support throughout this project. We also thank our peers for their feedback and our institution for providing essential resources that contributed to the successful completion of this project.

Abstract

This project presents a matrix calculator in assembly language, performing addition, subtraction, multiplication, and division. The program validates matrix dimensions before executing operations, providing accurate results. This low-level implementation demonstrates the practicality and efficiency of assembly in handling mathematical computations, offering a deeper understanding of matrix operations and computer architecture.

Project Overview

The matrix calculator is designed to perform essential matrix operations using assembly language. Users input two matrices and select an operation, with the program validating compatibility before displaying results. The project highlights matrix handling, error detection, and real-time computation, offering a hands-on application of linear algebra concepts, diverse environments, making it ideal for applications in robotics, automation, and education.

Introduction

Matrix operations are widely used in various fields, typically implemented in high-level languages for simplicity. This project takes on the challenge of implementing these operations in assembly language, combining concepts from

Linear Algebra and COAL courses. By developing this calculator, we deepen our understanding of both matrix computations and low-level programming.

Features:

1. **Matrix Dimension Validation:**

   Ensure that matrix dimensions are compatible for the selected operation (e.g., matching columns and rows for multiplication).

2. **Signed Integers Handling:**

   Supports operations on matrices with signed integers, correctly managing both positive and negative values during calculations.

3. **Error Handling:**

   Displays an error message if the chosen operation is not valid for the given matrices or if the dimensions are not valid for a particular operation.

4. **Matrix Addition:**

   Adds corresponding elements of two matrices and outputs the resultant matrix.

5. **Matrix Subtraction:**

   Subtracts elements of the second matrix from the first and displays the result.

6. **Matrix Multiplication:**

   Multiplies compatible matrices and outputs the resultant product matrix.

7. **<u>Matrix Transposition:</u>**

   Transposes the given matrix by swapping rows and columns, displaying the resulting matrix with dimensions reversed.

8. **<u>Result Display:</u>**

   Outputs the final computed matrix in a clear, formatted manner.

<u>Source Code:</u>

```
INCLUDE irvine32.inc

.data

; ############### DISPLAY VARIBALES ###############
Comma_seperator BYTE " , ",0
matrix_input_bracket BYTE "): ",0
spaces_to_print DWORD 10

Counter_row BYTE 1
Counter_col BYTE 1
entr_size_msg1 BYTE "Enter the rows in the Matrix 1: ",0
entr_size_msg2 BYTE "Enter the cols in the Matrix 1: ",0
entr_size_msg3 BYTE "Enter the rows in the Matrix 2: ",0
entr_size_msg4 BYTE "Enter the cols in the Matrix 2: ",0
display_msg1 BYTE "_______________ Matrix 1 _______________",10,10,0
display_msg2 BYTE "_______________ Matrix 2 _______________",10,10,0
display_msg3 BYTE "______________ Resultant ______________",10,0

element_counter_heading1 BYTE " (For Matrix 1) ------> Enter Element (",0
element_counter_heading2 BYTE " (For Matrix 2) ------> Enter Element (",0

msg1 BYTE "Matrix 1:", 0
msg2 BYTE "Matrix 2:", 0
msg3 BYTE "Transpose Result:", 0
msg4 BYTE "Addition Result:", 0
msg5 BYTE "Multiplication Result:", 0
msg6 BYTE "Subtraction Result:", 0
menu BYTE "Select an option:", 0
opt1 BYTE "1. Transpose Matrix 1", 0
opt2 BYTE "2. Add Matrix 1 and Matrix 2", 0
opt3 BYTE "3. Multiply Matrix 1 and Matrix 2", 0
```

```
opt2 BYTE "2. Add Matrix 1 and Matrix 2", 0
opt3 BYTE "3. Multiply Matrix 1 and Matrix 2", 0
opt4 BYTE "4. Sub Matrix 2 from Matrix 1", 0
invalid_inp BYTE "Invalid option, exiting program.", 0
miss_match_error BYTE "The dimentions of the matrix are not appropriate for this operation.", 0


transpose SDWORD 100 DUP(0)   ; For transposing a matrix
Matrix1 SDWORD 100 DUP (0)
Matrix2 SDWORD 100 DUP (0)
result SDWORD 100 DUP (0)
col_length1 DWORD 10
row_length1 DWORD 10
col_length2 DWORD 10
row_length2 DWORD 10
```

```
.code
; ########################### INPUT AND DISPLAY LOGIC ###########################
;-----------------------------------------
Take_input_Dimentions PROC
mov eax,0
mov edx , offset entr_size_msg1
Call WriteString
call readDec
mov col_length1,eax

mov edx , offset entr_size_msg2
Call WriteString
call readDec
mov row_length1,eax

mov edx , offset entr_size_msg3
Call WriteString
call readDec
mov col_length2,eax

mov edx , offset entr_size_msg4
Call WriteString
call readDec
mov row_length2,eax

call crlf
ret
Take_input_Dimentions endp
;-----------------------------------------
```

```
;---------------------------------------
print_nice_heading PROC heading_ptr:DWORD
movzx eax, Counter_row
mov edx, heading_ptr
call WriteString
call writeDec
mov edx, offset Comma_seperator
call WriteString
movzx eax, Counter_col
call writeDec
mov edx, offset matrix_input_bracket
call WriteString
call ReadInt
call crlf
ret
print_nice_heading endp
;---------------------------------------

;---------------------------------------
Take_elements_matrix_1 PROC
mov edi,0
mov esi,0
mov Counter_row,1
mov Counter_col,1

mov ecx, row_length1
row_input_loop:
push ecx
mov ecx, col_length1
col_input_loop:

```

```
invoke print_nice_heading, OFFSET element_counter_heading1
inc Counter_col

  mov Matrix1[esi*4],eax
  inc esi
  loop col_input_loop
pop ecx
inc Counter_row
mov counter_col,1
mov esi,0
add edi, col_length1
add esi, edi

loop row_input_loop
call crlf
ret
Take_elements_matrix_1 endp
;---------------------------------------

;---------------------------------------
Take_elements_matrix_2 PROC
mov edi,0
mov esi,0
mov Counter_row,1
mov Counter_col,1

mov ecx, row_length2
row_input_loop:
push ecx
mov ecx, col_length2
col_input_loop:
```

```
invoke print_nice_heading, OFFSET element_counter_heading2
inc Counter_col

  mov Matrix2[esi*4],eax
  inc esi
  loop col_input_loop
pop ecx
inc Counter_row
mov counter_col,1
mov esi,0
add edi, col_length2
add esi, edi

loop row_input_loop
call crlf
ret
Take_elements_matrix_2 endp
;----------------------------------------

;counting the number of digits in a number for clean output display
;------------------------------------------
CountDigits PROC USES eax ecx ; ebx contains number to count letters in

    mov spaces_to_print, 10
    mov eax, ebx
    mov ecx,1
    mov esi,10

    cmp eax,10
    jl countFinal
```

```
countLoop:
    cdq
    idiv esi                 ; Divide by 10
    cmp eax, 0
    je countFinal            ; If not zero, continue the loop
    inc ecx
    jmp countLoop

countFinal:
    sub spaces_to_print,ecx
    ret
CountDigits ENDP
;-------------------------------------------


;-----------------------------------------
printMatrix PROC USES eax ebx ecx edx esi,
    matrix_ptr:PTR SDWORD,
    printRows:SDWORD,
    printCols:SDWORD

    LOCAL row_index:SDWORD, col_index:SDWORD

    mov row_index, 0
row_loop:
    mov eax, row_index
    cmp eax, printRows
    jae done
    mov col_index, 0
```

```
 col_loop:
     mov eax, col_index
     cmp eax, printCols
     jae next_row

     ; Calculate offset: row_index * cols + col_index
     mov eax, row_index
     imul eax, printCols
     add eax, col_index
     shl eax, 2  ; Multiply by 4 for SDWORD

     ; Get matrix element and print it
     mov esi, matrix_ptr
     mov eax, [esi + eax]
     call WriteInt  ; Use WriteInt for signed integers

      push ecx
     mov ebx,eax
     mov eax, ' '
     call CountDigits
     mov ecx, spaces_to_print ;padding for nicer output
     Padding_loop:
     call WriteChar
     loop Padding_loop
     pop ecx

     inc col_index
     jmp col_loop
```

```
next_row:
    call Crlf
    inc row_index
    jmp row_loop
done:
    ret
printMatrix ENDP
;-----------------------------------------

; ########################### MATRICE OPERATIONS ###########################
;-----------------------------------------
transposeMatrix PROC USES eax ebx ecx edx esi edi,
    matrix_ptr:PTR SDWORD,
    transpose_ptr:PTR SDWORD,
    numRows:SDWORD,
    numCols:SDWORD

    LOCAL row:SDWORD, col:SDWORD

    mov row, 0
row_loop:
    ; Check if row is out of bounds
    mov eax, row
    cmp eax, numRows
    jae transpose_done

    mov col, 0
col_loop:
    ; Check if column is out of bounds
    mov eax, col
    cmp eax, numCols
```

```
    cmp eax, numCols
    jae next_row

    ; Calculate original matrix offset: row * numCols + col
    mov eax, row
    imul eax, numCols
    add eax, col
    shl eax, 2  ; Multiply by 4 (size of SDWORD)
    mov esi, matrix_ptr
    mov ebx, [esi + eax]

    ; Calculate transposed matrix offset: col * numRows + row
    mov eax, col
    imul eax, numRows
    add eax, row
    shl eax, 2
    mov edi, transpose_ptr
    mov [edi + eax], ebx

    inc col
    jmp col_loop

next_row:
    inc row
    jmp row_loop

transpose_done:
    ret
transposeMatrix ENDP
;----------------------------------------
```

```
;----------------------------------------
multiplyMatrix PROC USES eax ebx ecx edx esi edi,
    matrix1_ptr:PTR SDWORD,
    matrix2_ptr:PTR SDWORD,
    result_ptr:PTR SDWORD,
    rows_a:SDWORD,
    cols_a:SDWORD,
    cols_b:SDWORD

    LOCAL i:SDWORD, j:SDWORD, k:SDWORD, sum:SDWORD

    mov i, 0
outer_loop:
    mov eax, i
    cmp eax, rows_a
    jae multiply_done
    mov j, 0

middle_loop:
    mov eax, j
    cmp eax, cols_b
    jae next_row
    mov sum, 0
    mov k, 0

inner_loop:
    mov eax, k
    cmp eax, cols_a
    jae store_result

    ; Calculate matrix1 offset: i * cols_a + k
```

```
    ; Calculate matrix1 offset: i * cols_a + k
    mov eax, i
    imul eax, cols_a
    add eax, k
    shl eax, 2  ; Multiply by 4 for SDWORD

    mov esi, matrix1_ptr
    mov ebx, [esi + eax]
    push eax

    ; Calculate matrix2 offset: k * cols_b + j
    mov eax, k
    imul eax, cols_b
    add eax, j
    shl eax, 2

    mov esi, matrix2_ptr
    mov ecx, [esi + eax]
    pop eax

    imul ebx, ecx
    add sum, ebx

    inc k
    jmp inner_loop

store_result:
    mov eax, i
    imul eax, cols_b
    add eax, j
    shl eax, 2
```

```
    mov esi, result_ptr
    mov ebx, sum
    mov [esi + eax], ebx

    inc j
    jmp middle_loop

next_row:
    inc i
    jmp outer_loop

multiply_done:
    ret
multiplyMatrix ENDP
;-----------------------------------------

;-----------------------------------------
addMatrix PROC USES eax ebx ecx edx esi edi,
    matrix1_ptr:PTR SDWORD,
    matrix2_ptr:PTR SDWORD,
    result_ptr:PTR SDWORD,
    addRows:SDWORD,
    addCols:SDWORD

    LOCAL i:SDWORD, total_elements:SDWORD

    mov eax, addRows
    imul eax, addCols
    mov total_elements, eax
    mov i, 0

```

```
addition_loop:
    mov eax, i
    cmp eax, total_elements
    jae addition_done

    shl eax, 2  ; Multiply by 4 for SDWORD

    mov esi, matrix1_ptr
    mov ebx, [esi + eax]

    mov esi, matrix2_ptr
    mov ecx, [esi + eax]

    add ebx, ecx

    mov esi, result_ptr
    mov [esi + eax], ebx

    inc i
    jmp addition_loop

addition_done:
    ret
addMatrix ENDP
;----------------------------------------

;----------------------------------------
subMatrix PROC USES eax ebx ecx edx esi edi,
    matrix1_ptr:PTR SDWORD,
    matrix2_ptr:PTR SDWORD,
    result_ptr:PTR SDWORD,
```

```
    subRows:SDWORD,
    subCols:SDWORD

    LOCAL i:SDWORD, total_elements:SDWORD
    mov eax, subRows
    imul eax, subCols
    mov total_elements, eax

    mov i, 0

subtraction_loop:
    mov eax, i
    cmp eax, total_elements
    jae subtraction_done

    shl eax, 2

    mov esi, matrix1_ptr
    mov ebx, [esi + eax]

    mov esi, matrix2_ptr
    mov ecx, [esi + eax]

    sub ebx, ecx

    mov esi, result_ptr
    mov [esi + eax], ebx

    inc i
    jmp subtraction_loop

```

```
subtraction_done:
    ret
subMatrix ENDP
;----------------------------------------



;--------------------------------------------------------------------------------
main PROC
call Take_input_Dimentions
call Take_elements_matrix_1
call Take_elements_matrix_2

mov edx, offset display_msg1
call WriteString
INVOKE printMatrix, OFFSET Matrix1, row_length1, col_length1
call crlf
mov edx, offset display_msg2
call WriteString
INVOKE printMatrix, OFFSET Matrix2, row_length2, col_length2

call crlf
    lea edx, menu
    call WriteString
    call Crlf
    lea edx, opt1
    call WriteString
    call Crlf
    lea edx, opt2
    call WriteString
    call Crlf
```

```
    lea edx, opt3
    call WriteString
    call Crlf
    lea edx, opt4
    call WriteString
    call Crlf

    call ReadInt  ; Read user's choice
    mov ecx, eax  ; Store choice in ecx

    cmp ecx, 1
    je transpose_option

    cmp ecx, 2
    je addition_option

    cmp ecx, 3
    je multiplication_option

    cmp ecx, 4
    je subtraction_option

    call Crlf
    mov edx, offset invalid_inp
    call WriteString
    exit

    dimention_miss_match:
    mov edx, offset miss_match_error
    call WriteString
    exit
```

```
transpose_option:

    INVOKE transposeMatrix, OFFSET matrix1, OFFSET transpose, row_length1, col_length1

    call Crlf
    lea edx, msg3
    call WriteString
    call Crlf
    INVOKE printMatrix, OFFSET transpose, col_length1, row_length1
    jmp end_program

addition_option:

     mov eax, row_length1
     cmp eax, row_length2
     jne dimention_miss_match
     mov eax, col_length1
     cmp eax, col_length2
     jne dimention_miss_match

    INVOKE addMatrix, OFFSET matrix1, OFFSET matrix2, OFFSET result, row_length1, col_length1

    call Crlf
    lea edx, msg4
    call WriteString
    call Crlf
    INVOKE printMatrix, OFFSET result, row_length1, col_length1
    jmp end_program

```

```
multiplication_option:

     mov eax, col_length1
     cmp eax, row_length2
     jne dimention_miss_match
    INVOKE multiplyMatrix, OFFSET matrix1, OFFSET matrix2, OFFSET result, row_length1, col_length1, col_length2

    call Crlf
    lea edx, msg5
    call WriteString
    call Crlf
    INVOKE printMatrix, OFFSET result, row_length1, col_length2
    jmp end_program

subtraction_option:

     mov eax, row_length1
     cmp eax, row_length2
     jne dimention_miss_match
     mov eax, col_length1
     cmp eax, col_length2
     jne dimention_miss_match
    INVOKE subMatrix, OFFSET matrix1, OFFSET matrix2, OFFSET result, row_length1, col_length1

    call Crlf
    lea edx, msg6
    call WriteString
    call Crlf
    INVOKE printMatrix, OFFSET result, row_length1, col_length1
    jmp end_program

```

```
end_program:
exit
main endp
end main

;below is the code we made for dividing matrices with structures (handling fractions)

COMMENT!

INCLUDE Irvine32.inc

.data
slash BYTE "/",0
Comma_seperator BYTE " , ",0

Array_element STRUCT
    Numerator SDWORD ?
    Denominator SDWORD ?
Array_element ENDS

Temp_Operand_Arr1 Array_element <0,0>   ;copies operands and manupulates them for calculations
Temp_Operand_Arr2 Array_element <0,0>
Singular_Resultant Array_element <0,0>

First_Operand_Address DWORD 0
Second_Operand_Address DWORD 0

Array1 Array_element <1, 2>, <3, 4>, <-5, 6>, <7, 8>, <9, 10>
Array2 Array_element <5, 9>, <7, 3>, <2, 8>, <4, 6>, <10, 1>
col_length DWORD 5

```

```
Sign_flag BYTE 0

Smaller_value DWORD ?                  ;used in simplification function
temporary_Simplified_numerator DWORD ?    ;used in simplification function

.code

;Whenever you call this, have address of arrayu to display in ebx, along with col length in col_length
;-------------------------------------------------------------------------------
Display_Struct PROC
mov ecx, col_length

mov esi, 0
col_display_loop:
mov eax, [ebx + esi]
call writeInt
mov edx, offset slash
call writeString
mov eax, [ebx+ 4 +esi]
call WriteInt

mov edx, offset Comma_seperator
call writeString
add esi, sizeof Array_element
loop col_display_loop

ret
Display_Struct endp
;-------------------------------------------------------------------------------
```

```
;Utility function to copy 2 operands into a temporary variable to perform calculation
;------------------------------------------------------------------------------
Set_operands_for_calculations PROC  USES ecx ebx eax
mov ecx,First_Operand_Address
mov eax, [ecx]
lea ebx, Temp_Operand_Arr1
xchg eax, [ebx]


mov eax, [ecx+4]
lea ebx, Temp_Operand_Arr1
xchg eax, [ebx+4]

mov ecx,Second_Operand_Address
mov eax, [ecx]
lea ebx, Temp_Operand_Arr2
xchg eax, [ebx]

mov eax, [ecx+4]
lea ebx, Temp_Operand_Arr2
xchg eax, [ebx+4]
ret
Set_operands_for_calculations endp
;------------------------------------------------------------------------------



;whenever you call this, have the two operands address in the Operand_Address variables
;------------------------------------------------------------------------------
Divide_given_2_Elements PROC USES eax ebx ecx
mov Sign_flag,0
```

```
;whenever you call this, have the two operands address in the Operand_Address variables
;------------------------------------------------------------------------------
Divide_given_2_Elements PROC USES eax ebx ecx
mov Sign_flag,0

call Set_operands_for_calculations

lea ebx, Temp_Operand_Arr2
call Flip_Fraction                ;convert (a/b)/(c/d) to a/b * d/c

lea ebx, Temp_Operand_Arr1
lea eax, Temp_Operand_Arr2
mov ecx, [eax]

mov eax, [ebx]
imul ecx
lea edx, Singular_Resultant      ;move result of multiplication into Singular_Resultant variable
xchg [edx], eax

lea eax, Temp_Operand_Arr2+4
mov ecx, [eax]    ;now multiply denominators
mov eax, [ebx+4]
imul ecx
lea edx, Singular_Resultant      ;move result of multiplication into Singular_Resultant variable
xchg [edx+4], eax

lea ebx,Singular_Resultant[0]       ;here, if resultant is signed, remove sign for simplification
mov eax,[ebx]
cmp eax,0                               ; handling case -a/b  or a/-b or a\b
jge Not_signed_NUM

```

```
;####################################      ;case -a/b
mov Sign_flag,1
mov ecx, -1
imul ecx
xchg [ebx], eax
jmp Not_signed_DENOM                    ; answer can never be -a/-b so dont check denom
;####################################


;####################################
Not_signed_NUM:

mov eax, [ebx+4]
cmp eax,0
jge Not_signed_DENOM
                                          ;case -a/b
mov Sign_flag,1
imul ecx
lea ebx,Singular_Resultant[0]
xchg [ebx+4], eax
;####################################

Not_signed_DENOM:                          ;case a/b
call Simplify_Fraction


movzx ecx, Sign_flag                 ;if result was signed before simplification, restore sign.
cmp ecx,1
jne Was_not_signed_before

mov ecx,-1
lea ebx,Singular_Resultant[0]
mov eax,[ebx]
imul ecx
xchg [ebx], eax


Was_not_signed_before:
ret
Divide_given_2_Elements endp
;-------------------------------------------------------------------------------


;whenever you call this, have the address of element to flip in ebx
;-------------------------------------------------------------------------------
Flip_Fraction PROC USES eax
mov eax, [ebx]
xchg [ebx+4], eax
xchg eax, [ebx]
ret
Flip_Fraction endp
;-------------------------------------------------------------------------------

;-------------------------------------------------------------------------------
Get_smaller_value PROC    USES eax
lea ebx, Singular_Resultant
mov eax, [ebx]
cmp eax, [ebx+4]              ;equal case to be handled
je Set_to_1
jg Denom_is_smaller
jmp skip_rest_of_cases    ; eax already has numerator

```

```
Denom_is_smaller:
mov eax, [ebx+4]
jmp skip_rest_of_cases

Set_to_1:
mov eax, 1
xchg [ebx], eax
mov eax,1
xchg [ebx+4], eax
ret

skip_rest_of_cases:
mov Smaller_value, eax
ret
Get_smaller_value endp
;--------------------------------------------------------------------------------



;before calling this, make sure the fraction to simplify is in 'Singular_Resultant' variable
;--------------------------------------------------------------------------------
Simplify_Fraction PROC USES ebx edi eax edx

lea ebx, Singular_Resultant
mov edi,2

Simplification_comparision_loop:
call Get_smaller_value
mov edx,0
cmp edi, Smaller_value
jg end_simplification
```

```
mov eax, [ebx]
idiv edi
cmp edx,0
jne Cant_divide_here
mov temporary_Simplified_numerator,eax ;save numerator

mov edx,0
mov eax, [ebx+4]
idiv edi
cmp edx,0
jne Cant_divide_here

xchg [ebx+4],eax
mov eax, temporary_Simplified_numerator
xchg [ebx], eax
mov edi,1                                ; reset edi if division happened

Cant_divide_here:
inc edi
jmp Simplification_comparision_loop

end_simplification:
ret
Simplify_Fraction endp
;--------------------------------------------------------------------------------
```

```
main PROC
lea ebx, Array1
call Display_Struct
call crlf
add ebx, 16
mov First_Operand_Address, ebx    ;4th index struct1 is divisor

lea ebx, Array2
call Display_Struct
call crlf
add ebx, 16
mov Second_Operand_Address, ebx ;4th index struct1 is dividant

call Divide_given_2_Elements
lea ebx, Singular_Resultant
mov eax, [ebx]
call WriteInt
mov edx, offset slash
call WriteString
mov eax, [ebx+4]
call WriteInt

exit
main endp
end main
!
```

Output

1. Matrix Transposition

```
Enter the rows in the Matrix 1: 3
Enter the cols in the Matrix 1: 3
Enter the rows in the Matrix 2: 3
Enter the cols in the Matrix 2: 3

 (For Matrix 1) ------> Enter Element (1 , 1): 1

 (For Matrix 1) ------> Enter Element (1 , 2): 2

 (For Matrix 1) ------> Enter Element (1 , 3): 3

 (For Matrix 1) ------> Enter Element (2 , 1): 4

 (For Matrix 1) ------> Enter Element (2 , 2): 5

 (For Matrix 1) ------> Enter Element (2 , 3): 6

 (For Matrix 1) ------> Enter Element (3 , 1): 7

 (For Matrix 1) ------> Enter Element (3 , 2): 8

 (For Matrix 1) ------> Enter Element (3 , 3): 9


 (For Matrix 2) ------> Enter Element (1 , 1): 2

 (For Matrix 2) ------> Enter Element (1 , 2): 4

 (For Matrix 2) ------> Enter Element (1 , 3): 5

 (For Matrix 2) ------> Enter Element (2 , 1): 8

 (For Matrix 2) ------> Enter Element (2 , 2): 15

 (For Matrix 2) ------> Enter Element (2 , 3): 33

 (For Matrix 2) ------> Enter Element (3 , 1): 7
```

```
 (For Matrix 2) ------> Enter Element (3 , 1): 7

 (For Matrix 2) ------> Enter Element (3 , 2): 34

 (For Matrix 2) ------> Enter Element (3 , 3): 53


_______________ Matrix 1 _______________

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

_______________ Matrix 2 _______________

+2          +4          +5
+8          +15         +33
+7          +34         +53

Select an option:
1. Transpose Matrix 1
2. Add Matrix 1 and Matrix 2
3. Multiply Matrix 1 and Matrix 2
4. Sub Matrix 2 from Matrix 1
1

Transpose Result:
+1          +4          +7
+2          +5          +8
+3          +6          +9
```

2. Matrix Addition

```
Enter the rows in the Matrix 1: 3
Enter the cols in the Matrix 1: 3
Enter the rows in the Matrix 2: 3
Enter the cols in the Matrix 2: 3

 (For Matrix 1) ------> Enter Element (1 , 1): 4

 (For Matrix 1) ------> Enter Element (1 , 2): 3

 (For Matrix 1) ------> Enter Element (1 , 3): 2

 (For Matrix 1) ------> Enter Element (2 , 1): 5

 (For Matrix 1) ------> Enter Element (2 , 2): 17

 (For Matrix 1) ------> Enter Element (2 , 3): 35

 (For Matrix 1) ------> Enter Element (3 , 1): 23

 (For Matrix 1) ------> Enter Element (3 , 2): 24

 (For Matrix 1) ------> Enter Element (3 , 3): 2


 (For Matrix 2) ------> Enter Element (1 , 1): 23

 (For Matrix 2) ------> Enter Element (1 , 2): 56

 (For Matrix 2) ------> Enter Element (1 , 3): 3

 (For Matrix 2) ------> Enter Element (2 , 1): 2

 (For Matrix 2) ------> Enter Element (2 , 2): 4

 (For Matrix 2) ------> Enter Element (2 , 3): 98

 (For Matrix 2) ------> Enter Element (3 , 1): 2

 (For Matrix 2) ------> Enter Element (3 , 2): 3

 (For Matrix 2) ------> Enter Element (3 , 3): 5
```

```
_______________ Matrix 1 _______________

+4          +3          +2
+5          +17         +35
+23         +24         +2

_______________ Matrix 2 _______________

+23         +56         +3
+2          +4          +98
+2          +3          +5

Select an option:
1. Transpose Matrix 1
2. Add Matrix 1 and Matrix 2
3. Multiply Matrix 1 and Matrix 2
4. Sub Matrix 2 from Matrix 1
2

Addition Result:
+27         +59         +5
+7          +21         +133
+25         +27         +7
```

3. Matrix Subtraction

```
Enter the rows in the Matrix 1: 3
Enter the cols in the Matrix 1: 3
Enter the rows in the Matrix 2: 3
Enter the cols in the Matrix 2: 3

 (For Matrix 1) ------> Enter Element (1 , 1): 1

 (For Matrix 1) ------> Enter Element (1 , 2): 3

 (For Matrix 1) ------> Enter Element (1 , 3): 5

 (For Matrix 1) ------> Enter Element (2 , 1): 63

 (For Matrix 1) ------> Enter Element (2 , 2): 2

 (For Matrix 1) ------> Enter Element (2 , 3): 5

 (For Matrix 1) ------> Enter Element (3 , 1): 3

 (For Matrix 1) ------> Enter Element (3 , 2): 2

 (For Matrix 1) ------> Enter Element (3 , 3): 5


 (For Matrix 2) ------> Enter Element (1 , 1): 7

 (For Matrix 2) ------> Enter Element (1 , 2): 2

 (For Matrix 2) ------> Enter Element (1 , 3): 44

 (For Matrix 2) ------> Enter Element (2 , 1): 23

 (For Matrix 2) ------> Enter Element (2 , 2): 25

 (For Matrix 2) ------> Enter Element (2 , 3): 4

 (For Matrix 2) ------> Enter Element (3 , 1): 7

 (For Matrix 2) ------> Enter Element (3 , 2): 2

 (For Matrix 2) ------> Enter Element (3 , 3): 1
```

```
_______________ Matrix 1 _______________

+1          +3          +5
+63         +2          +5
+3          +2          +5

_______________ Matrix 2 _______________

+7          +2          +44
+23         +25         +4
+7          +2          +1

Select an option:
1. Transpose Matrix 1
2. Add Matrix 1 and Matrix 2
3. Multiply Matrix 1 and Matrix 2
4. Sub Matrix 2 from Matrix 1
4

Subtraction Result:
-6          +1          -39
+40         -23          +1
-4          +0          +4
```

4. Matrix Multiplication

```
Enter the rows in the Matrix 1: 3
Enter the cols in the Matrix 1: 3
Enter the rows in the Matrix 2: 3
Enter the cols in the Matrix 2: 3

 (For Matrix 1) ------> Enter Element (1 , 1): 23

 (For Matrix 1) ------> Enter Element (1 , 2): 21

 (For Matrix 1) ------> Enter Element (1 , 3): 56

 (For Matrix 1) ------> Enter Element (2 , 1): 3

 (For Matrix 1) ------> Enter Element (2 , 2): 3

 (For Matrix 1) ------> Enter Element (2 , 3): 2

 (For Matrix 1) ------> Enter Element (3 , 1): 5

 (For Matrix 1) ------> Enter Element (3 , 2): 2

 (For Matrix 1) ------> Enter Element (3 , 3): 76


 (For Matrix 2) ------> Enter Element (1 , 1): 12

 (For Matrix 2) ------> Enter Element (1 , 2): 3

 (For Matrix 2) ------> Enter Element (1 , 3): 76

 (For Matrix 2) ------> Enter Element (2 , 1): 43

 (For Matrix 2) ------> Enter Element (2 , 2): 45

 (For Matrix 2) ------> Enter Element (2 , 3): 75

 (For Matrix 2) ------> Enter Element (3 , 1): 3

 (For Matrix 2) ------> Enter Element (3 , 2): 21

 (For Matrix 2) ------> Enter Element (3 , 3): 3
```

```
____________________ Matrix 1 ____________________

+23         +21         +56
+3          +3          +2
+5          +2          +76

____________________ Matrix 2 ____________________

+12         +3          +76
+43         +45         +75
+3          +21         +3

Select an option:
1. Transpose Matrix 1
2. Add Matrix 1 and Matrix 2
3. Multiply Matrix 1 and Matrix 2
4. Sub Matrix 2 from Matrix 1
3

Multiplication Result:
+1347       +2190       +3491
+171        +186        +459
+374        +1701       +758
```

5 Matrix Division

NOTE: division typically isn’t defined in matrices, so for division, we made the prototype using structures and fractions in a different program

(Code given in comment below the project)

for a hardcoded array: (dividing element 3 of matrix 1 with matrix 2)

```
Microsoft Visual Studio Debug Console
+1/+2 , +3/+4 , -5/+6 , +7/+8 , +9/+10 ,
+5/+9 , +7/+3 , +2/+8 , +4/+6 , +10/+1 ,
-10/+3
```

$$((-5/6) / (2/8)) = -5/6 * 8/2 = -40/12 = -10/3$$

The division handles fractions, signed values, and simplification of the fractions using the following:

-) A structure that contains numerator and denominator

-) A function to Flip a fraction (convert division into multiplication)

-) A function to simplify the resultant fraction