

# **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. Matrix Transposition:

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

## 8. Result Display:

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

## Source Code:

```
1  INCLUDE irvine32.inc
2
3  .data
4
5  ; ##### DISPLAY VARIABLES #####
6  Comma_seperator BYTE " , ",0
7  matrix_input_bracket BYTE ")": ",0
8  spaces_to_print DWORD 10
9
10 Counter_row BYTE 1
11 Counter_col BYTE 1
12 entr_size_msg1 BYTE "Enter the rows in the Matrix 1: ",0
13 entr_size_msg2 BYTE "Enter the cols in the Matrix 1: ",0
14 entr_size_msg3 BYTE "Enter the rows in the Matrix 2: ",0
15 entr_size_msg4 BYTE "Enter the cols in the Matrix 2: ",0
16 display_msg1 BYTE "----- Matrix 1 -----",10,10,0
17 display_msg2 BYTE "----- Matrix 2 -----",10,10,0
18 display_msg3 BYTE "----- Resultant -----",10,0
19
20 element_counter_heading1 BYTE " (For Matrix 1) -----> Enter Element ",0
21 element_counter_heading2 BYTE " (For Matrix 2) -----> Enter Element ",0
22
23 msg1 BYTE "Matrix 1:", 0
24 msg2 BYTE "Matrix 2:", 0
25 msg3 BYTE "Transpose Result:", 0
26 msg4 BYTE "Addition Result:", 0
27 msg5 BYTE "Multiplication Result:", 0
28 msg6 BYTE "Subtraction Result:", 0
29 menu BYTE "Select an option:", 0
30 opt1 BYTE "1. Transpose Matrix 1", 0
31 opt2 BYTE "2. Add Matrix 1 and Matrix 2", 0
32 opt3 BYTE "3. Multiply Matrix 1 and Matrix 2", 0
```

```

31 opt2 BYTE "2. Add Matrix 1 and Matrix 2", 0
32 opt3 BYTE "3. Multiply Matrix 1 and Matrix 2", 0
33 opt4 BYTE "4. Sub Matrix 2 from Matrix 1", 0
34 invalid_inp BYTE "Invalid option, exiting program.", 0
35 miss_match_error BYTE "The dimentions of the matrix are not appropriate for this operation.", 0
36
37
38 transpose SDWORD 100 DUP(0) ; For transposing a matrix
39 Matrix1 SDWORD 100 DUP (0)
40 Matrix2 SDWORD 100 DUP (0)
41 result SDWORD 100 DUP (0)
42 col_length1 DWORD 10
43 row_length1 DWORD 10
44 col_length2 DWORD 10
45 row_length2 DWORD 10
46

```

```

47 .code
48 ; ##### INPUT AND DISPLAY LOGIC #####
49 ;-----
50 Take_input_Dimention PROC
51 mov eax,0
52 mov edx , offset entr_size_msg1
53 Call WriteString
54 call readDec
55 mov col_length1,eax
56
57 mov edx , offset entr_size_msg2
58 Call WriteString
59 call readDec
60 mov row_length1,eax
61
62 mov edx , offset entr_size_msg3
63 Call WriteString
64 call readDec
65 mov col_length2,eax
66
67 mov edx , offset entr_size_msg4
68 Call WriteString
69 call readDec
70 mov row_length2,eax
71
72 call crlf
73 ret
74 Take_input_Dimention endp
75 ;-----
76

```

```

78 ;-----
79 print_nice_heading PROC heading_ptr:DWORD
80 movzx eax, Counter_row
81 mov edx, heading_ptr
82 call WriteString
83 call writeDec
84 mov edx, offset Comma_seperator
85 call WriteString
86 movzx eax, Counter_col
87 call writeDec
88 mov edx, offset matrix_input_bracket
89 call WriteString
90 call ReadInt
91 call crlf
92 ret
93 print_nice_heading endp
94 ;-----
95
96 ;-----
97 Take_elements_matrix_1 PROC
98 mov edi,0
99 mov esi,0
100 mov Counter_row,1
101 mov Counter_col,1
102
103 mov ecx, row_length1
104 row_input_loop:
105 push ecx
106 mov ecx, col_length1
107 col_input_loop:
108

```

```

109 invoke print_nice_heading, OFFSET element_counter_heading1
110 inc Counter_col
111
112 mov Matrix1[esi*4],eax
113 inc esi
114 loop col_input_loop
115 pop ecx
116 inc Counter_row
117 mov counter_col,1
118 mov esi,0
119 add edi, col_length1
120 add esi, edi
121
122 loop row_input_loop
123 call crlf
124 ret
125 Take_elements_matrix_1 endp
126 ;-----
127
128 ;-----
129 Take_elements_matrix_2 PROC
130 mov edi,0
131 mov esi,0
132 mov Counter_row,1
133 mov Counter_col,1
134
135 mov ecx, row_length2
136 row_input_loop:
137 push ecx
138 mov ecx, col_length2
139 col_input_loop:

```

```

141 invoke print_nice_heading, OFFSET element_counter_heading2
142 inc Counter_col
143
144     mov Matrix2[esi*4],eax
145     inc esi
146     loop col_input_loop
147 pop ecx
148 inc Counter_row
149 mov counter_col,1
150 mov esi,0
151 add edi, col_length2
152 add esi, edi
153
154 loop row_input_loop
155 call crlf
156 ret
157 Take_elements_matrix_2 endp
158 ;-----
159
160 ;counting the number of digits in a number for clean output display
161 ;-----
162 CountDigits PROC USES eax ecx ; ebx contains number to count letters in
163
164     mov spaces_to_print, 10
165     mov eax, ebx
166     mov ecx,1
167     mov esi,10
168
169     cmp eax,10
170     jl countFinal
171

```

```

172 countLoop:
173     cdq
174     idiv esi ; Divide by 10
175     cmp eax, 0
176     je countFinal ; If not zero, continue the loop
177     inc ecx
178     jmp countLoop
179
180 countFinal:
181     sub spaces_to_print,ecx
182     ret
183 CountDigits ENDP
184 ;-----
185
186 ;-----
187 printMatrix PROC USES eax ebx ecx edx esi,
188     matrix_ptr:PTR SDWORD,
189     printRows:SDWORD,
190     printCols:SDWORD
191
192     LOCAL row_index:SDWORD, col_index:SDWORD
193
194     mov row_index, 0
195 row_loop:
196     mov eax, row_index
197     cmp eax, printRows
198     jae done
199     mov col_index, 0
200

```

```

202     col_loop:
203         mov eax, col_index
204         cmp eax, printCols
205         jae next_row
206
207         ; Calculate offset: row_index * cols + col_index
208         mov eax, row_index
209         imul eax, printCols
210         add eax, col_index
211         shl eax, 2 ; Multiply by 4 for SDWORD
212
213         ; Get matrix element and print it
214         mov esi, matrix_ptr
215         mov eax, [esi + eax]
216         call WriteInt ; Use WriteInt for signed integers
217
218         push ecx
219         mov ebx, eax
220         mov eax, ' '
221         call CountDigits
222         mov ecx, spaces_to_print ;padding for nicer output
223         Padding_loop:
224         call WriteChar
225         loop Padding_loop
226         pop ecx
227
228         inc col_index
229         jmp col_loop
230

```

```

231     next_row:
232         call Crlf
233         inc row_index
234         jmp row_loop
235     done:
236         ret
237     printMatrix ENDP
238     ;-----
239
240     ; ##### MATRICE OPERATIONS #####
241     ;-----
242     transposeMatrix PROC USES eax ebx ecx edx esi edi,
243         matrix_ptr:PTR SDWORD,
244         transpose_ptr:PTR SDWORD,
245         numRows:SDWORD,
246         numCols:SDWORD
247
248     LOCAL row:SDWORD, col:SDWORD
249
250     mov row, 0
251     row_loop:
252         ; Check if row is out of bounds
253         mov eax, row
254         cmp eax, numRows
255         jae transpose_done
256
257         mov col, 0
258     col_loop:
259         ; Check if column is out of bounds
260         mov eax, col
261         cmp eax, numCols

```



```

261         cmp eax, numCols
262         jae next_row
263
264         ; Calculate original matrix offset: row * numCols + col
265         mov eax, row
266         imul eax, numCols
267         add eax, col
268         shl eax, 2 ; Multiply by 4 (size of SDWORD)
269         mov esi, matrix_ptr
270         mov ebx, [esi + eax]
271
272         ; Calculate transposed matrix offset: col * numRows + row
273         mov eax, col
274         imul eax, numRows
275         add eax, row
276         shl eax, 2
277         mov edi, transpose_ptr
278         mov [edi + eax], ebx
279
280         inc col
281         jmp col_loop
282
283     next_row:
284         inc row
285         jmp row_loop
286
287     transpose_done:
288         ret
289     transposeMatrix ENDP
290 ;-----

```

```

292 ;-----
293 multiplyMatrix PROC USES eax ebx ecx edx esi edi,
294     matrix1_ptr:PTR SDWORD,
295     matrix2_ptr:PTR SDWORD,
296     result_ptr:PTR SDWORD,
297     rows_a:SDWORD,
298     cols_a:SDWORD,
299     cols_b:SDWORD
300
301     LOCAL i:SDWORD, j:SDWORD, k:SDWORD, sum:SDWORD
302
303     mov i, 0
304     outer_loop:
305         mov eax, i
306         cmp eax, rows_a
307         jae multiply_done
308         mov j, 0
309
310     middle_loop:
311         mov eax, j
312         cmp eax, cols_b
313         jae next_row
314         mov sum, 0
315         mov k, 0
316
317     inner_loop:
318         mov eax, k
319         cmp eax, cols_a
320         jae store_result
321
322         ; Calculate matrix1 offset: i * cols_a + k

```

```
322     ; Calculate matrix1 offset: i * cols_a + k
323     mov eax, i
324     imul eax, cols_a
325     add eax, k
326     shl eax, 2 ; Multiply by 4 for SDWORD
327
328     mov esi, matrix1_ptr
329     mov ebx, [esi + eax]
330     push eax
331
332     ; Calculate matrix2 offset: k * cols_b + j
333     mov eax, k
334     imul eax, cols_b
335     add eax, j
336     shl eax, 2
337
338     mov esi, matrix2_ptr
339     mov ecx, [esi + eax]
340     pop eax
341
342     imul ebx, ecx
343     add sum, ebx
344
345     inc k
346     jmp inner_loop
347
348 store_result:
349     mov eax, i
350     imul eax, cols_b
351     add eax, j
352     shl eax, 2
```

```

354         mov esi, result_ptr
355         mov ebx, sum
356         mov [esi + eax], ebx
357
358         inc j
359         jmp middle_loop
360
361     next_row:
362         inc i
363         jmp outer_loop
364
365     multiply_done:
366         ret
367     multiplyMatrix ENDP
368     ;-----
369
370     ;-----
371     addMatrix PROC USES eax ebx ecx edx esi edi,
372         matrix1_ptr:PTR SDWORD,
373         matrix2_ptr:PTR SDWORD,
374         result_ptr:PTR SDWORD,
375         addRows:SDWORD,
376         addCols:SDWORD
377
378     LOCAL i:SDWORD, total_elements:SDWORD
379
380         mov eax, addRows
381         imul eax, addCols
382         mov total_elements, eax
383         mov i, 0
384

```

```

385     addition_loop:
386         mov eax, i
387         cmp eax, total_elements
388         jae addition_done
389
390         shl eax, 2 ; Multiply by 4 for SDWORD
391
392         mov esi, matrix1_ptr
393         mov ebx, [esi + eax]
394
395         mov esi, matrix2_ptr
396         mov ecx, [esi + eax]
397
398         add ebx, ecx
399
400         mov esi, result_ptr
401         mov [esi + eax], ebx
402
403         inc i
404         jmp addition_loop
405
406     addition_done:
407         ret
408     addMatrix ENDP
409     ;-----
410
411     ;-----
412     subMatrix PROC USES eax ebx ecx edx esi edi,
413         matrix1_ptr:PTR SDWORD,
414         matrix2_ptr:PTR SDWORD,
415         result_ptr:PTR SDWORD,

```

```
416     subRows:SDWORD,  
417     subCols:SDWORD  
418  
419     LOCAL i:SDWORD, total_elements:SDWORD  
420     mov eax, subRows  
421     imul eax, subCols  
422     mov total_elements, eax  
423  
424     mov i, 0  
425  
426 subtraction_loop:  
427     mov eax, i  
428     cmp eax, total_elements  
429     jae subtraction_done  
430  
431     shl eax, 2  
432  
433     mov esi, matrix1_ptr  
434     mov ebx, [esi + eax]  
435  
436     mov esi, matrix2_ptr  
437     mov ecx, [esi + eax]  
438  
439     sub ebx, ecx  
440  
441     mov esi, result_ptr  
442     mov [esi + eax], ebx  
443  
444     inc i  
445     jmp subtraction_loop  
446
```

```

447 subtraction_done:
448     ret
449 subMatrix ENDP
450 ;-----
451
452
453
454 ;-----
455 main PROC
456     call Take_input_Dimensions
457     call Take_elements_matrix_1
458     call Take_elements_matrix_2
459
460     mov edx, offset display_msg1
461     call WriteString
462     INVOKE printMatrix, OFFSET Matrix1, row_length1, col_length1
463     call Crlf
464     mov edx, offset display_msg2
465     call WriteString
466     INVOKE printMatrix, OFFSET Matrix2, row_length2, col_length2
467
468     call Crlf
469     lea edx, menu
470     call WriteString
471     call Crlf
472     lea edx, opt1
473     call WriteString
474     call Crlf
475     lea edx, opt2
476     call WriteString
477     call Crlf

```

```

478     lea edx, opt3
479     call WriteString
480     call Crlf
481     lea edx, opt4
482     call WriteString
483     call Crlf
484
485     call ReadInt ; Read user's choice
486     mov ecx, eax ; Store choice in ecx
487
488     cmp ecx, 1
489     je transpose_option
490
491     cmp ecx, 2
492     je addition_option
493
494     cmp ecx, 3
495     je multiplication_option
496
497     cmp ecx, 4
498     je subtraction_option
499
500     call Crlf
501     mov edx, offset invalid_inp
502     call WriteString
503     exit
504
505     dimension_miss_match:
506     mov edx, offset miss_match_error
507     call WriteString
508     exit

```

```

510 transpose_option:
511
512     INVOKE transposeMatrix, OFFSET matrix1, OFFSET transpose, row_length1, col_length1
513
514     call Crlf
515     lea edx, msg3
516     call WriteString
517     call Crlf
518     INVOKE printMatrix, OFFSET transpose, col_length1, row_length1
519     jmp end_program
520
521 addition_option:
522
523     mov eax, row_length1
524     cmp eax, row_length2
525     jne dimention_miss_match
526     mov eax, col_length1
527     cmp eax, col_length2
528     jne dimention_miss_match
529
530     INVOKE addMatrix, OFFSET matrix1, OFFSET matrix2, OFFSET result, row_length1, col_length1
531
532     call Crlf
533     lea edx, msg4
534     call WriteString
535     call Crlf
536     INVOKE printMatrix, OFFSET result, row_length1, col_length1
537     jmp end_program
538

```

```

539 multiplication_option:
540
541     mov eax, col_length1
542     cmp eax, row_length2
543     jne dimention_miss_match
544     INVOKE multiplyMatrix, OFFSET matrix1, OFFSET matrix2, OFFSET result, row_length1, col_length1, col_length2
545
546     call Crlf
547     lea edx, msg5
548     call WriteString
549     call Crlf
550     INVOKE printMatrix, OFFSET result, row_length1, col_length2
551     jmp end_program
552
553 subtraction_option:
554
555     mov eax, row_length1
556     cmp eax, row_length2
557     jne dimention_miss_match
558     mov eax, col_length1
559     cmp eax, col_length2
560     jne dimention_miss_match
561     INVOKE subMatrix, OFFSET matrix1, OFFSET matrix2, OFFSET result, row_length1, col_length1
562
563     call Crlf
564     lea edx, msg6
565     call WriteString
566     call Crlf
567     INVOKE printMatrix, OFFSET result, row_length1, col_length1
568     jmp end_program
569

```

```

570     end_program:
571     exit
572     main endp
573     end main
574
575     ;below is the code we made for dividing matrices with structures (handling fractions)
576
577     COMMENT!
578
579     INCLUDE Irvine32.inc
580
581     .data
582     slash BYTE "/",0
583     Comma_seperator BYTE " , ",0
584
585     Array_element STRUCT
586         Numerator SDWORD ?
587         Denominator SDWORD ?
588     Array_element ENDS
589
590     Temp_Operand_Arr1 Array_element <0,0> ;copies operands and manipulates them for calculations
591     Temp_Operand_Arr2 Array_element <0,0>
592     Singular_Resultant Array_element <0,0>
593
594     First_Operand_Address DWORD 0
595     Second_Operand_Address DWORD 0
596
597     Array1 Array_element <1, 2>, <3, 4>, <-5, 6>, <7, 8>, <9, 10>
598     Array2 Array_element <5, 9>, <7, 3>, <2, 8>, <4, 6>, <10, 1>
599     col_length DWORD 5
600

```

```

601     Sign_flag BYTE 0
602
603     Smaller_value DWORD ? ;used in simplification function
604     temporary_Simplified_numerator DWORD ? ;used in simplification function
605
606     .code
607
608     ;Whenever you call this, have address of arrayu to display in ebx, along with col length in col_length
609     ;-----
610     Display_Struct PROC
611     mov ecx, col_length
612
613     mov esi, 0
614     col_display_loop:
615     mov eax, [ebx + esi]
616     call writeInt
617     mov edx, offset slash
618     call writeString
619     mov eax, [ebx+ 4 +esi]
620     call WriteInt
621
622     mov edx, offset Comma_seperator
623     call writeString
624     add esi, sizeof Array_element
625     loop col_display_loop
626
627     ret
628     Display_Struct endp
629     ;-----
630

```



```

633 ;Utility function to copy 2 operands into a temporary variable to perform calculation
634 ;-----
635 Set_operands_for_calculations PROC USES ecx ebx eax
636 mov ecx,First_Operand_Address
637 mov eax,[ecx]
638 lea ebx,Temp_Operand_Arr1
639 xchg eax,[ebx]
640
641
642 mov eax,[ecx+4]
643 lea ebx,Temp_Operand_Arr1
644 xchg eax,[ebx+4]
645
646 mov ecx,Second_Operand_Address
647 mov eax,[ecx]
648 lea ebx,Temp_Operand_Arr2
649 xchg eax,[ebx]
650
651 mov eax,[ecx+4]
652 lea ebx,Temp_Operand_Arr2
653 xchg eax,[ebx+4]
654 ret
655 Set_operands_for_calculations endp
656 ;-----
657
658
659
660 ;whenever you call this, have the two operands address in the Operand_Address variables
661 ;-----
662 Divide_given_2_Elements PROC USES eax ebx ecx
663 mov Sign_flag,0

```

```

660 ;whenever you call this, have the two operands address in the Operand_Address variables
661 ;-----
662 Divide_given_2_Elements PROC USES eax ebx ecx
663 mov Sign_flag,0
664
665 call Set_operands_for_calculations
666
667 lea ebx,Temp_Operand_Arr2
668 call Flip_Fraction ;convert (a/b)/(c/d) to a/b * d/c
669
670 lea ebx,Temp_Operand_Arr1
671 lea eax,Temp_Operand_Arr2
672 mov ecx,[eax]
673
674 mov eax,[ebx]
675 imul ecx
676 lea edx,Singular_Resultant ;move result of multiplication into Singular_Resultant variable
677 xchg [edx],eax
678
679 lea eax,Temp_Operand_Arr2+4
680 mov ecx,[eax] ;now multiply denominators
681 mov eax,[ebx+4]
682 imul ecx
683 lea edx,Singular_Resultant ;move result of multiplication into Singular_Resultant variable
684 xchg [edx+4],eax
685
686 lea ebx,Singular_Resultant[0] ;here, if resultant is signed, remove sign for simplification
687 mov eax,[ebx]
688 cmp eax,0 ; handling case -a/b or a/-b or a\b
689 jge Not_signed_NUM
690

```

```

691 ;##### ;case -a/b
692 mov Sign_flag,1
693 mov ecx, -1
694 imul ecx
695 xchg [ebx], eax
696 jmp Not_signed_DENOM ; answer can never be -a/-b so dont check denom
697 ;#####
698
699
700 ;#####
701 Not_signed_NUM:
702
703 mov eax, [ebx+4]
704 cmp eax,0
705 jge Not_signed_DENOM
706
707 ;case -a/b
708 mov Sign_flag,1
709 imul ecx
710 lea ebx,Singular_Resultant[0]
711 xchg [ebx+4], eax
712 ;#####
713 Not_signed_DENOM: ;case a/b
714 call Simplify_Fraction
715
716
717 movzx ecx, Sign_flag ;if result was signed before simplification, restore sign.
718 cmp ecx,1
719 jne Was_not_signed_before
720
721 mov ecx,-1

```

```

722 lea ebx,Singular_Resultant[0]
723 mov eax,[ebx]
724 imul ecx
725 xchg [ebx], eax
726
727
728 Was_not_signed_before:
729 ret
730 Divide_given_2_Elements endp
731 ;-----
732
733
734 ;whenever you call this, have the address of element to flip in ebx
735 ;-----
736 Flip_Fraction PROC USES eax
737 mov eax, [ebx]
738 xchg [ebx+4], eax
739 xchg eax, [ebx]
740 ret
741 Flip_Fraction endp
742 ;-----
743
744 ;-----
745 Get_smaller_value PROC USES eax
746 lea ebx, Singular_Resultant
747 mov eax, [ebx]
748 cmp eax, [ebx+4] ;equal case to be handled
749 je Set_to_1
750 jg Denom_is_smaller
751 jmp skip_rest_of_cases ; eax already has numerator
752

```

```

753 Denom_is_smaller:
754 mov eax, [ebx+4]
755 jmp skip_rest_of_cases
756 |
757 Set_to_1:
758 mov eax, 1
759 xchg [ebx], eax
760 mov eax, 1
761 xchg [ebx+4], eax
762 ret
763
764 skip_rest_of_cases:
765 mov Smaller_value, eax
766 ret
767 Get_smaller_value endp
768 ;-----
769
770
771
772 ;before calling this, make sure the fraction to simplify is in 'Singular_Resultant' variable
773 ;-----
774 Simplify_Fraction PROC USES ebx edi eax edx
775
776 lea ebx, Singular_Resultant
777 mov edi, 2
778
779 Simplification_comparision_loop:
780 call Get_smaller_value
781 mov edx, 0
782 cmp edi, Smaller_value
783 jg end_simplification

```

```

784 mov eax, [ebx]
785 idiv edi
786 cmp edx, 0
787 jne Cant_divide_here
788 mov temporary_Simplified_numerator, eax ;save numerator
789
790 mov edx, 0
791 mov eax, [ebx+4]
792 idiv edi
793 cmp edx, 0
794 jne Cant_divide_here
795
796 xchg [ebx+4], eax
797 mov eax, temporary_Simplified_numerator
798 xchg [ebx], eax
799 mov edi, 1 ; reset edi if division happened
800
801 Cant_divide_here:
802 inc edi
803 jmp Simplification_comparision_loop
804
805 end_simplification:
806 ret
807 Simplify_Fraction endp
808 ;-----

```

```
810     main PROC
811     lea ebx, Array1
812     call Display_Struct
813     call crlf
814     add ebx, 16
815     mov First_Operand_Address, ebx    ;4th index struct1 is divisor
816
817     lea ebx, Array2
818     call Display_Struct
819     call crlf
820     add ebx, 16
821     mov Second_Operand_Address, ebx ;4th index struct1 is dividant
822
823     call Divide_given_2_Elements
824     lea ebx, Singular_Resultant
825     mov eax, [ebx]
826     call WriteInt
827     mov edx, offset slash
828     call WriteString
829     mov eax, [ebx+4]
830     call WriteInt
831
832     exit
833     main endp
834     end main
835     !
```

## 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

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:

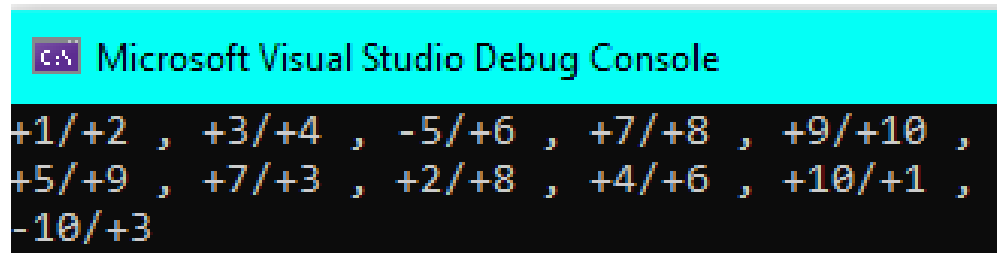
+1347	+2190	+3491
+171	+186	+459
+374	+1701	+758

## 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 )

A screenshot of the Microsoft Visual Studio Debug Console. The title bar is blue and says "Microsoft Visual Studio Debug Console". The console output shows a list of fractions: +1/+2 , +3/+4 , -5/+6 , +7/+8 , +9/+10 , +5/+9 , +7/+3 , +2/+8 , +4/+6 , +10/+1 , -10/+3. The text is in a monospaced font with a light blue background.

```
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