****

**UNIVERSITY OF KARACHI**

**DEPARTMENT OF COMPUTER SCIENCE**

**(MORNING PROGRAMME) MASTERS IN COMPUTER SCIENCE (FINAL) FIRST SEMESTER 2020**

**CS-623 DATA STRUCTURE AND ALGORITHM ANALYSIS**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**GRAND ASSIGNMENT II**

**2D-ARRAY**

**GROUP MEMBERS**

|  |  |
| --- | --- |
| **STUDENT-ID** | **NAME OF STUDENT** |
| **P1810020** | **HIRA HASSAN AWAN** |
| **P1810033** | **NABEEL AHMED** |
| **P1810050** | **SUMBUL ZOHRAB** |
| **P1810055** | **SYEDA ELIA BATOOL** |

2-DIMENSIONAL arrays

*Definition:*

The **two**-**dimensional array** can be **defined** as an **array** of **arrays**. It is the simplest form of multidimensional array. A 2-dimensional array is a list of one-dimensional arrays. However, **2D arrays** are created to implement a relational database lookalike data structure.

The **2D array**  (two dimensional) array is also known as matrix. A matrix can be represented as a table of rows and columns. Thus, every element in the array **A** is identified by an element name of the form **A[i,j]**

Where **A** is the name of the array, and **i** and **j** are subscripts that uniquely identify each element in the array A.

Syntax:

Data\_type[ , ] array\_name = new Data\_Type[ Row, Column];

Initialization of 2-D array

Multidimensional arrays may be initialized by specifying bracketed values for each row. The following array is with 2 rows and each row has 4 columns.

There are two ways to initialize a two Dimensional arrays during declaration.

Int arr [2] [4] = {

{1,2,3,4}, {5,6,7,8}

} ;

OR

Int arr [2] [4] = {1,2,3,4,5,6,7,8,} ;

When we initialize a normal array (or you can say one dimensional array) during declaration, we need not to specify the size of it. However that’s not the case with 2D array, you must always specify the second dimension even if you are specifying elements during the declaration.

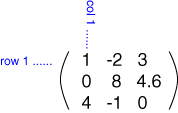
This creates a 2D array of int that has 8 elements arranged in 2 rows and 4 columns. For example, this statement creates the 2-by-4 array that is shown below:

int[ ][ ] A = { { 1, 2, 3, 4 },

{ 5, 6, 7, 8 } };

Matrix:

A **matrix** is a rectangular array of numbers or other mathematical objects for which operations such as addition and multiplication are defined. It is a collection of numbers arranged into a fixed number of rows and columns. Usually the numbers are real numbers. In general, matrices can contain complex numbers but we won’t see those here. For example, a matrix with three rows and three columns:



This matrix is a 3x3 matrix because it has three rows and three columns. In describing matrices, the format is:

Rows X Columns

Each number that makes up a row is called an element of the matrix. The elements in a matrix have specific locations.

**Types of Matrix**

* Null matrix
* Identity matrix
* Diagonal matrix
* Symmetric matrix
* Orthogonal matix
* Scalar matrix

Symmetric matrix:

A matrix whose transpose is the same as the original matrix is called a **symmetric matrix**. A symmetric matrix must be a square matrix. Only a square matrix can be a symmetric matrix.

Example:

1 2 3 1 4 7

4 5 6 = 2 5 8

7 8 9 3 6 9

In terms of its elements, if a matrix A is symmetric, then

A(i,j) = A(j,i)

Diagonal matrix:

A **diagonal matrix** is a square matrix where all the elements are 0 except for those in the diagonal from the top left corner to the bottom right corner.

Example:

1 0 0 2 0 0 0

0 5 0 OR 0 8 0 0

0 0 9 0 0 6 0

0 0 0 5

Identity matrix:

A square **matrix** in which all the main diagonal elements are 1's and all the remaining elements are 0's is called an **Identity Matrix**. **Identity Matrix** is also called Unit **Matrix** or Elementary **Matrix.**

Example

1 0 0 1 0 0 0

0 1 0 OR 0 1 0 0

0 0 1 0 0 1 0

0 0 0 1

**Operations on Matrix**

* Addition
* Subtraction
* Multiplication
* Inverse
* Transpose
* Rotate
* Rotate Up
* Rotate Down
* Rotate Right
* Rotate Left
* Face Rotation

**Matrix ADDITION**

If two matrices have same number of rows and same number of columns, then the matrix sum can be computed.

If A is a MxN matrix formed by adding corresponding elements of A and B.

Example:

1 2 3 1 4 7

A = 4 5 6 + B = 2 5 8

7 8 9 3 6 9

2 6 10

A+B = 6 10 14

10 14 18

**Matrix SUBTRACTION:**

If A and B have the same number of rows and columns,then:

A-B is defined as A + (-B).

To Compute A-B, subtract the corresponding element of B from each element of A.

Example:

1 2 3 1 4 7

A = 4 5 6 - B = 2 5 8

7 8 9 3 6 9

0 -2 -4

A-B = 2 0 -2

4 2 0

**Matrix MULTIPLICATION:**

For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix. The result matrix, known as the **matrix product**, has the number of rows of the first and the number of columns of the second matrix.

To multiply matrices, you'll need to multiply the elements (or numbers) in the row of the first matrix by the elements in the rows of the second matrix and add their products. You can multiply matrices in just a few easy steps that require addition, multiplication, and the proper placement of the results.

Example:

A = -1 4 x B = 9 -3

2 3 6 1

= (-1)(9)+(4)(6) (-1)(-3)+(4)(1)

(2)(9)+(3)(6) (2)(-3)+(3)(1)

A x B = 15 7

36 -3

**TRANSPOSE:**

The transpose of a matrix is a new matrix whose rows are the columns of the original. This makes the columns of new matrix the rows of the original.

Example:

1 2 3 T 1 4 7

4 5 6 = 2 5 8

7 8 9 3 6 9

**MENU OF 2D ARRAY:**

1-CREATE

2-INSERT

3-UPDATE

4-DELETE

5-OUTPUT

6-ROTATE

7-OPERATION

8-STATISTICS

**CREATE:**

1. Start
2. Read logrow
3. Read logcol
4. Call check

Else

For i=1 to logrow then

For j=1 to logrcol then

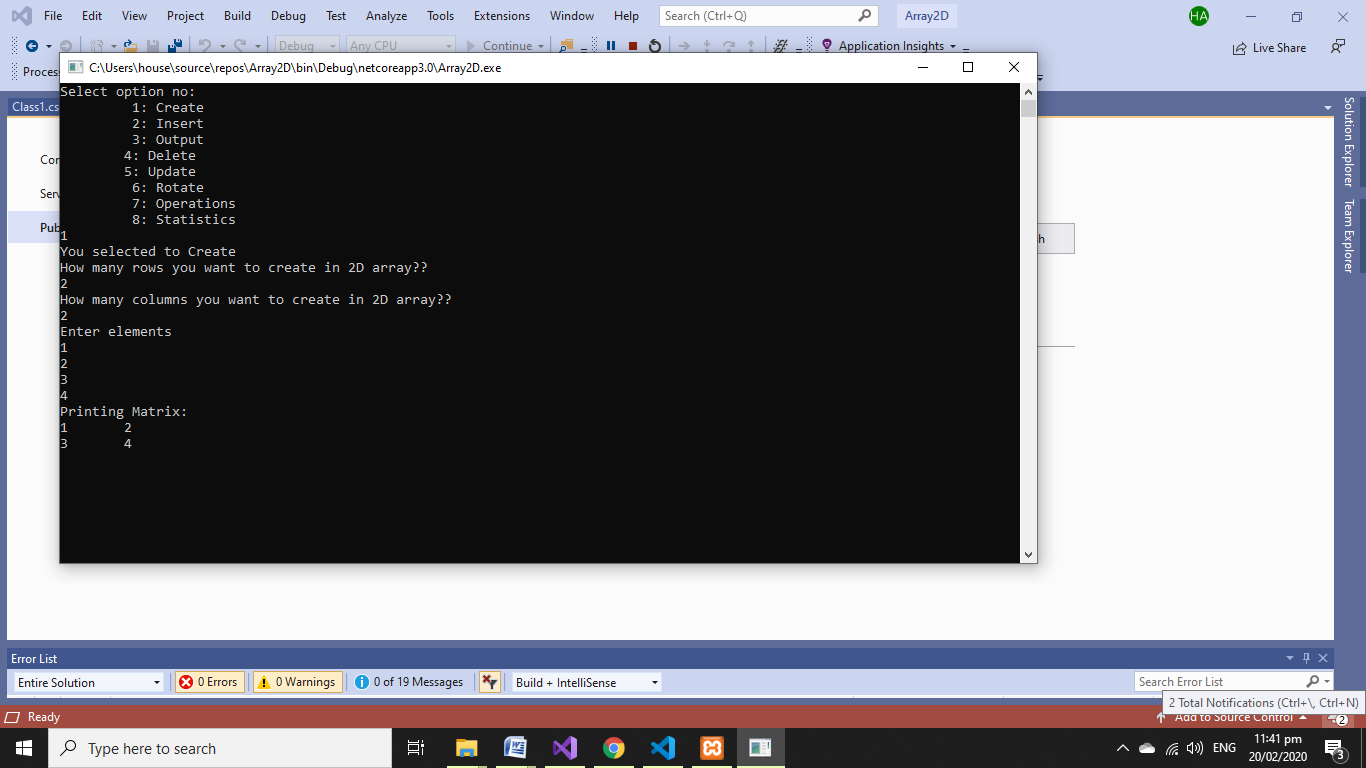
Read arrI,j

( end of j loop)

( end of i loop)

( end of call check)

1. Write arrI,j
2. Exit.

****

**INSERT:**

1. Start
2. Call check

Read choice

1. Check choice ==1

Read row number

Inclogrow

1. For i=log row DOWNTO rownum THEN

For j=1 to logcol

Read arrI,j

( end of i loop)

( end of i loop)

( end of call check)

1. Else if choice ==2

Read col number

InclogcolS

1. For i=column to rowsize

Write arrI,j

1. For i=1 to phyrow Then

For j=1 to phycolThen

Read arrI,j

( end of i loop)

( end of i loop)

( end of call check)

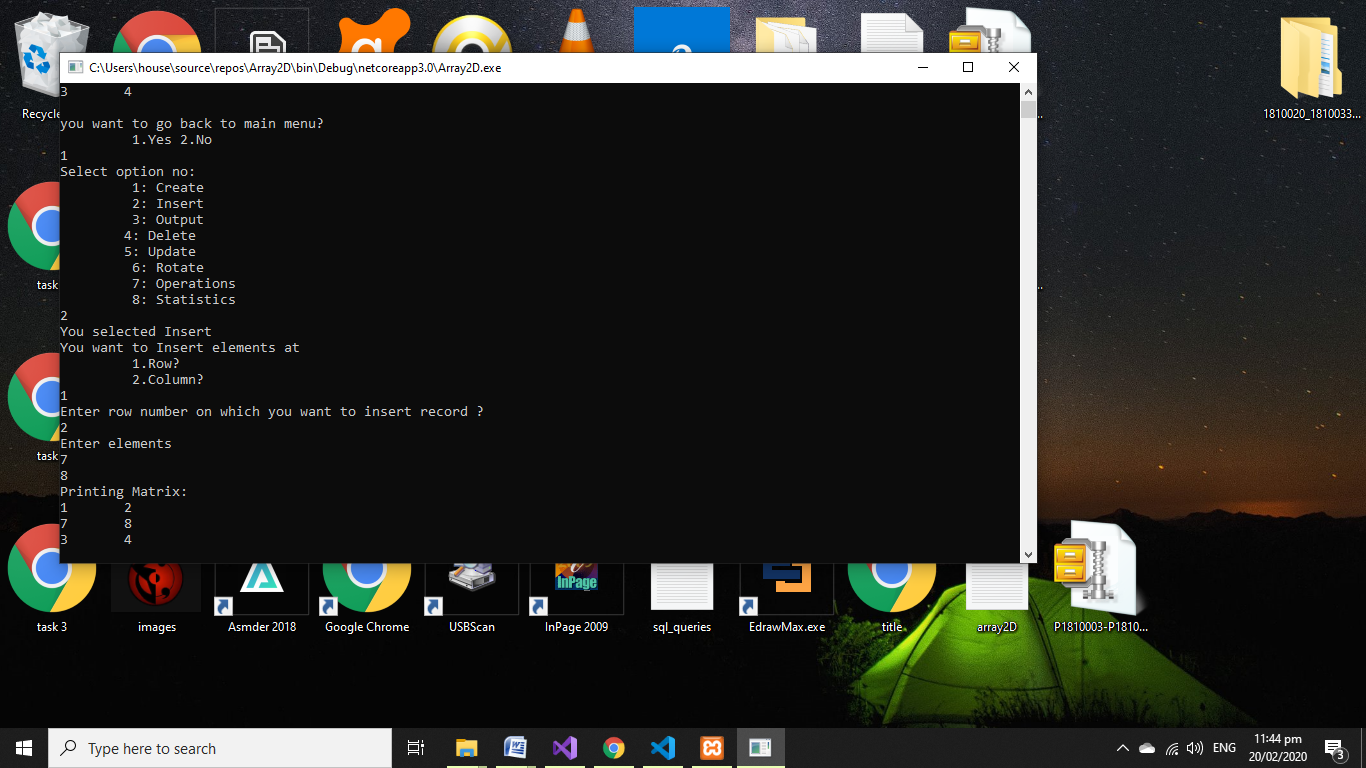
1. Write Invalid option
2. CHECK

Else if logrow>=phyrow OR logcol>phycol

Write limit is reached

(End of else if)

1. Exit

****

**UPDATE:**

1. Start
2. CHECK

If logrow>=0 $$ logcol>=0 Goto

(end of if)

1. CHECK if upchoice== 1 goto

Read posrow

1. For i=1 to logrow Then

For j=1 to j<= logcol Then

If i<posrow

Write arrI,j

(end of if )

( end of j loop)

( end of i loop)

Else if i==posrow

Read arrI,j

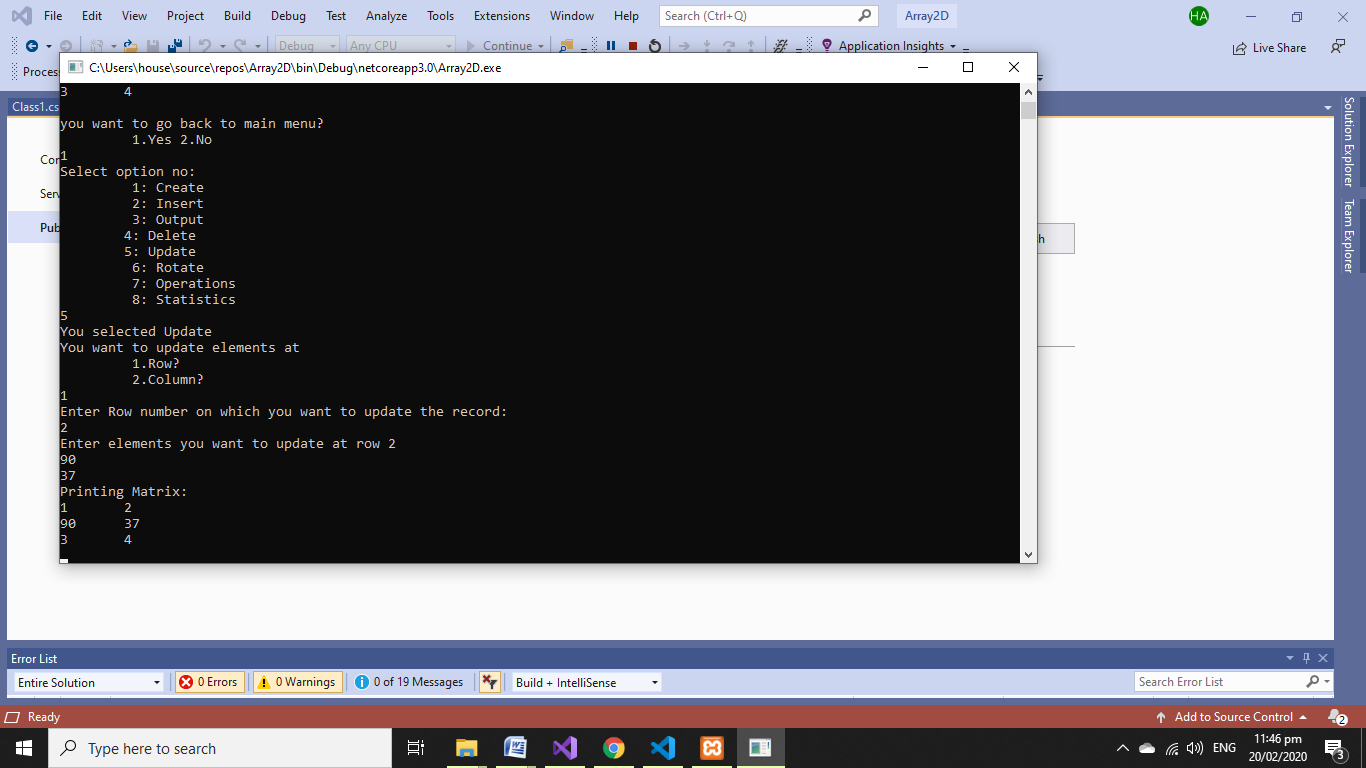
(end of else if)

Else

ArrI,j = arrI,j

(end of else)

1. Exit

****

**DELETE:**

1. Start
2. Check If logrow>=0 $$ logcol>=0 Goto

Read dechoice

(end of if )

1. Check dechoice== 1 Goto

Read Rowpos

1. Check Rowpos>rowsizeGoto

Write Deletion is not possible

(end of if)

else

For i=1 to logrow Then

For j=1 to j<= logcol Then

If i<rowpos

Write arrI,j

(end of if )

( end of j loop)

( end of i loop)

Else

ArrI,j= arr (i+1,j)

(end of if statement step 3)

1. Check

Else if dechoice == 2

1. Read colpos
2. Check

If colpos>colsize

Write delete is not possible

(end of if )

Else

For i=1 to log col Then

For j=1 to logrow then

If j<colpos

Write arrI,j

(end of if )

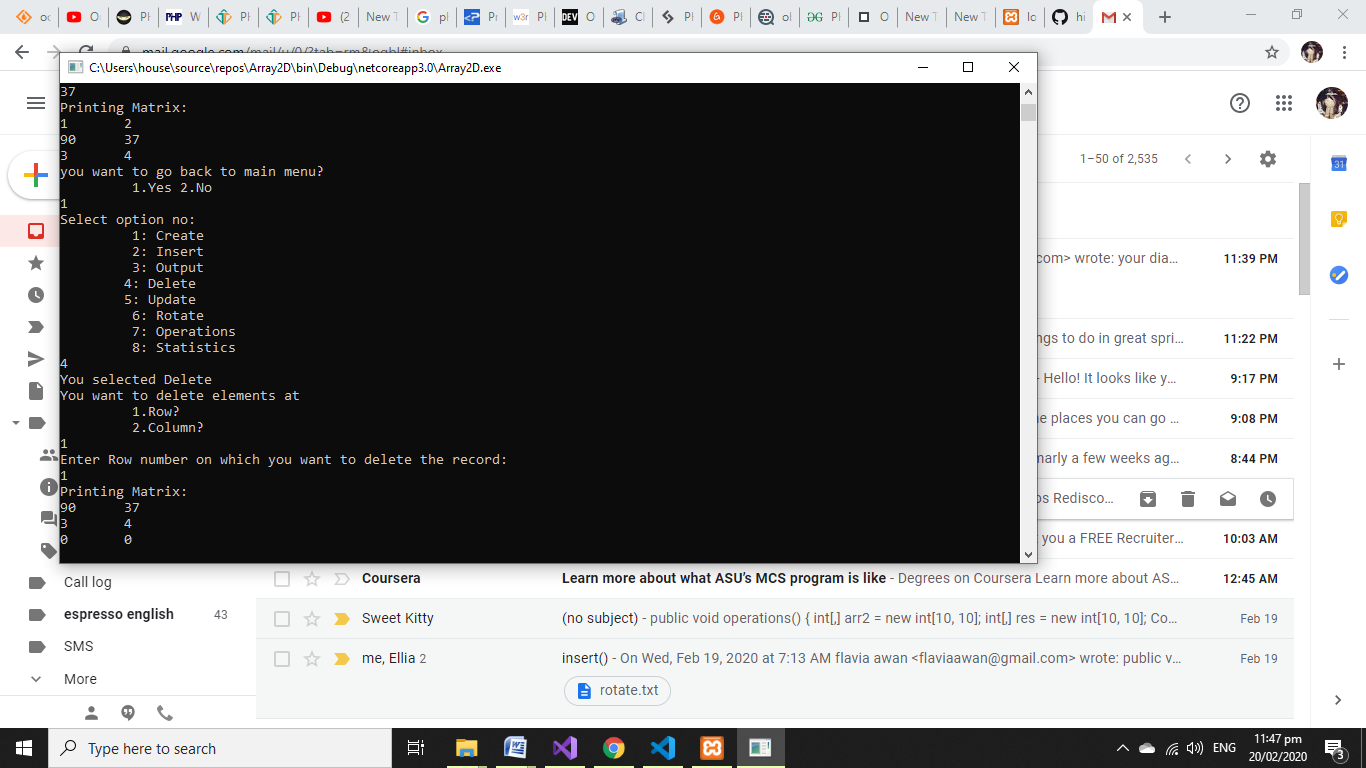
( end of j loop)

( end of i loop)

Else

Write arr(I,j+1)

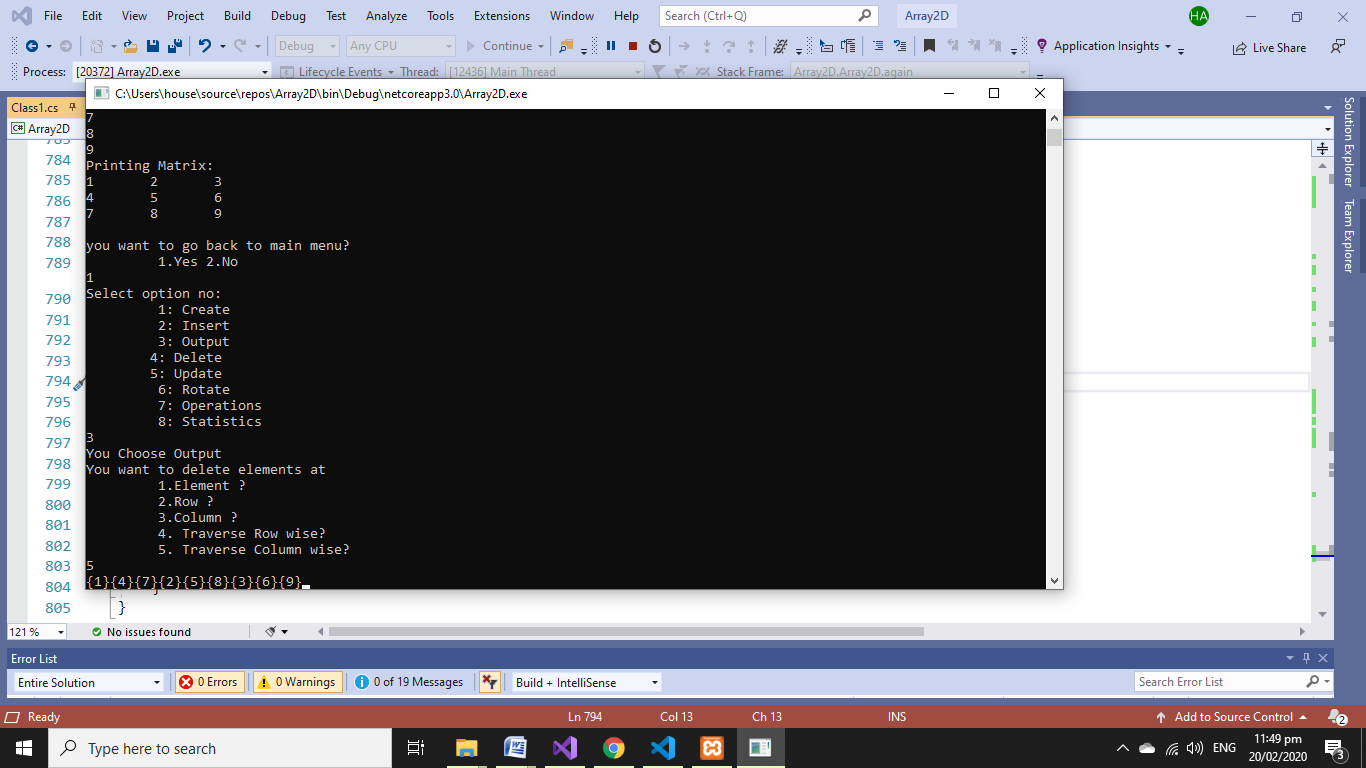
1. Exit.

****

**OUTPUT:**

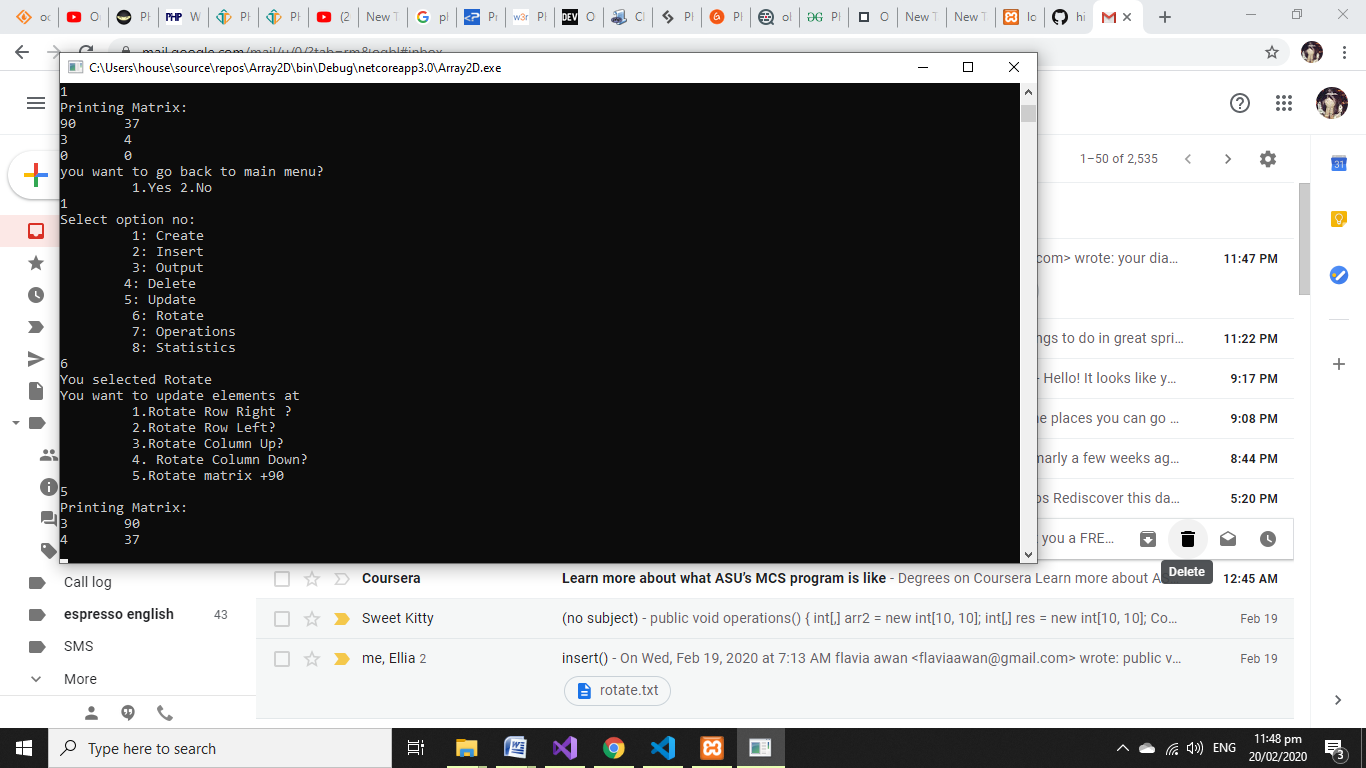
**Start**

**Exit**

****

**ROTATE:**

1. Start
2. Procedure Rotate:
3. Call procedure check
4. Check:
5. (logrow >= 0 && logcol >= 0)
6. temp🡨0
7. Read choice
8. Check:
9. If choice==1 // rotate row right
10. Then read rownum
11. temp 🡨 arr[rownum,logcol]
12. i 🡨 rownum
13. for j🡨logcol-1 down to 1
14. arr[rownum, j + 1] 🡨 arr[rownum, j]
15. end of j
16. arr[rownum, 1] 🡨 temp
17. call procedure print
18. end if
19. check:
20. if choice==2 // rotate row left
21. Then read rownum
22. temp 🡨 arr[rownum,1]
23. i 🡨 rownum
24. for j🡨1 down to logcol
25. arr[rownum, j ] 🡨 arr[rownum, j+1]
26. end of j
27. arr[rownum, logcol] 🡨 temp
28. call procedure print
29. end if
30. check:
31. if choice==3 // rotate column up
32. Then read colnum
33. temp 🡨 arr[1,colnum]
34. j 🡨 colnum
35. for i🡨1 to logrow
36. arr[i,colnum ] 🡨 arr[i+1,colnum]
37. end of i
38. arr[logrow, colnum] 🡨 temp
39. call procedure print
40. end if
41. check:
42. if choice==3 // rotate column up
43. Then read colnum
44. temp 🡨 arr[logrow,colnum]
45. j 🡨 colnum
46. for i🡨logrow-1 down to 1
47. arr[i+1,colnum ] 🡨 arr[i,colnum]
48. end of i
49. arr[1, colnum] 🡨 temp
50. call procedure print
51. end if
52. end if
53. Exit



**OPERATIONS:**

1. Start
2. Read choice
3. If choice🡨1
4. Call procedure Addition
5. If choice🡨2
6. Call procedure Subtration
7. If choice🡨3
8. Call Procedure Multiplication
9. Exit

## ALGORITHM OF ADDITION

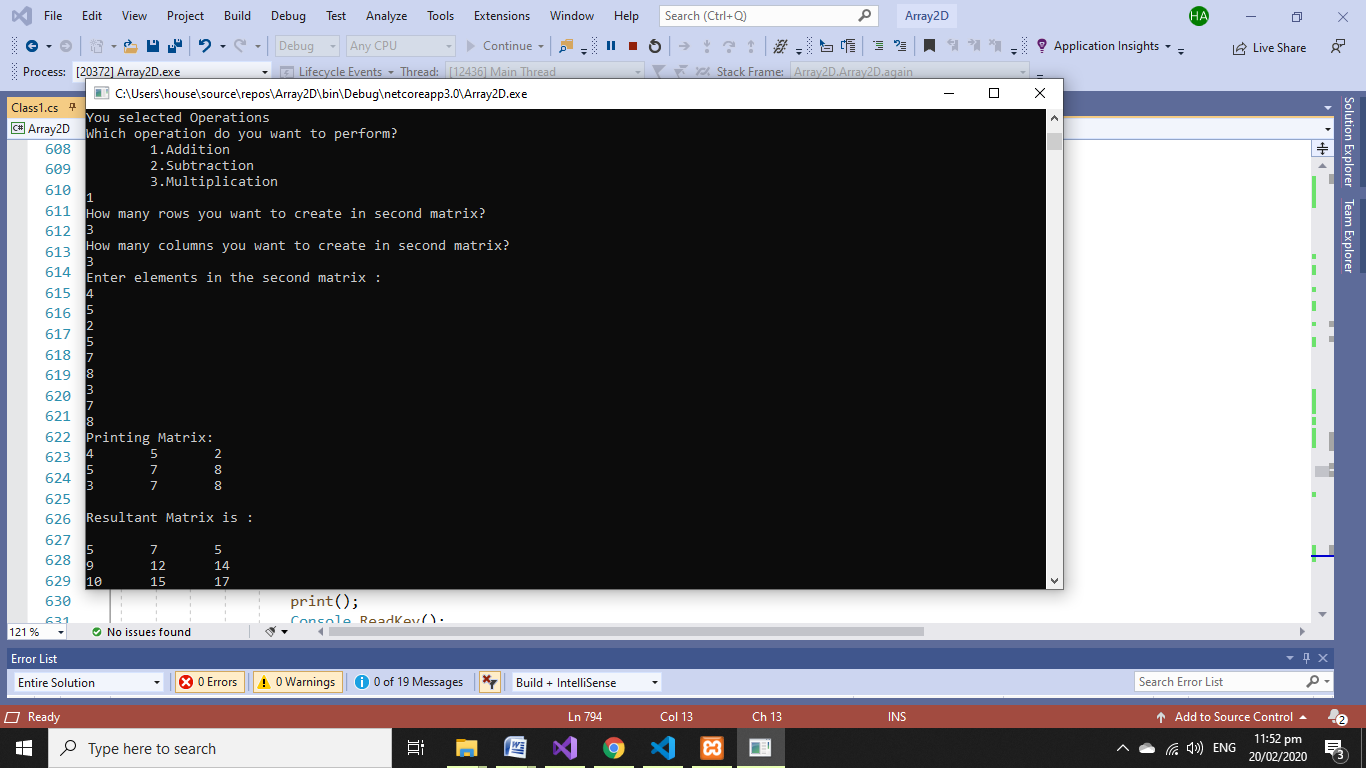
* Input the order of the matrix.
* Input the matrix 1 elements.
* Input the matrix 2 elements.
* Repeat from i = 0 to m
* Repeat from j = 0 to n
* mat3[i][j] = mat1[i][j] + mat2[i][j]
* Print mat3

## ALGORITHM OF SUBTRACTION

* Input the order of the matrix.
* Input the matrix 1 elements.
* Input the matrix 2 elements.
* Repeat from i = 0 to m
* Repeat from j = 0 to n
* mat3[i][j] = mat1[i][j] – mat2[i][j]
* Print mat3

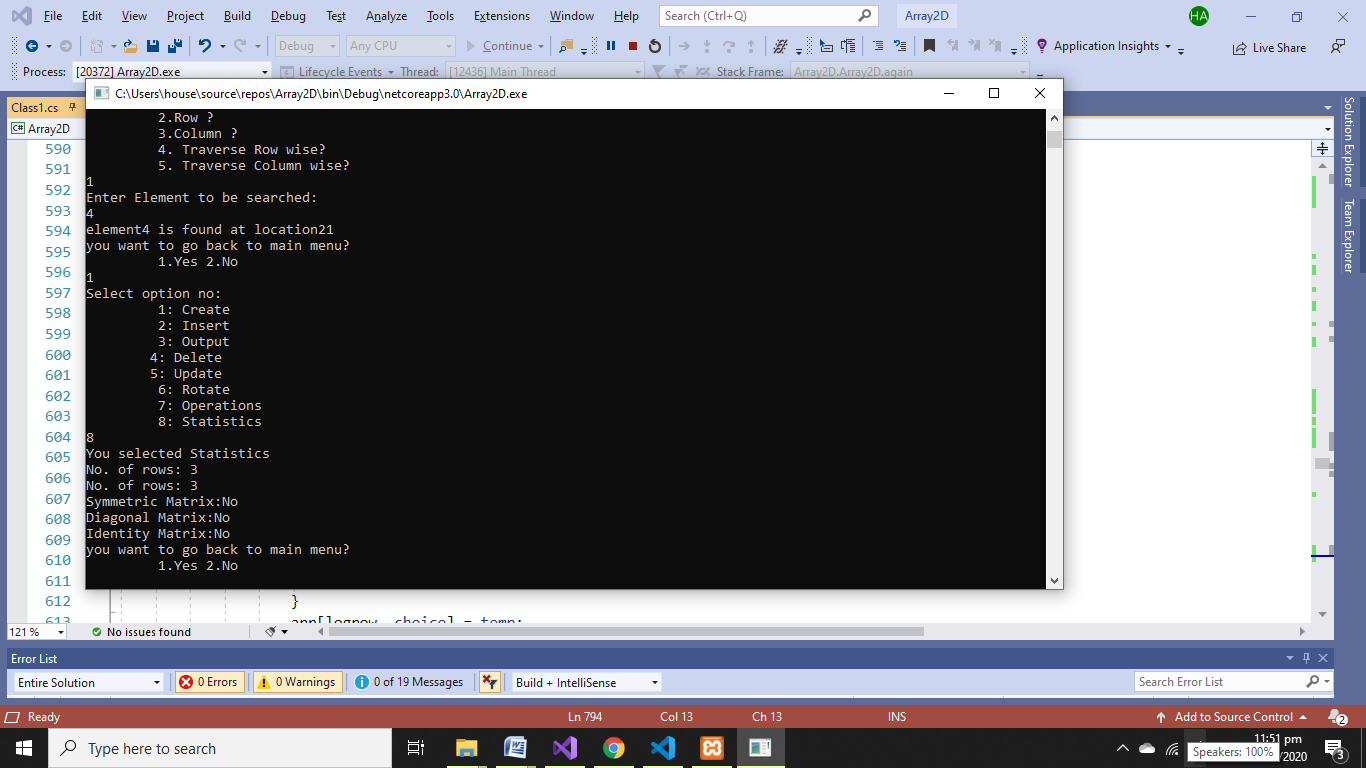
## ALGORITHM OF MULTIPLICATION

* Input the order of the matrix1 ( m \* n).
* Input the order of matrix2 (p \* q).
* Input the matrix 1 elements.
* Input the matrix 2 elements
* Repeat from i = 0 to m
* Repeat from j = 0 to q
* repeat from k = 0 to p
* sum = sum + mat1[c][k] \* mat2[k][d];
* mat3[c][d] = sum
* Print mat3.

****

**STATISTICS:**

1. Start
2. Write logrow
3. Write logcol
4. Return procedure Symmetric
5. Return procedure Diagonal
6. Return procedure Identity
7. Exit

****

**FUNCTIONAL FLOW DIAGRAM:**

**Main Menu**

Main Menu

**Rotate()**

**Insert()**

**Update()**

**Create()**e

**Delete()**

**Output() ()**

**Operation()**

**Statistics()**

**print()**

**again()**

**check()**

**Addition()**

**Subtraction()**

**Symmetrical()**

**Multiplication()**

**Diagonal()**

90

**Identity()**

**printResult()**

**Transpose()**

**THE END**

**x------x-----x------x**