Objective:

- Resolving issues related to pointer as data member.
- Focusing on Object's initialization and resource Allocation/de-allocation issues and issues related to object manipulation.
- And a bit of logic as always to keep your brains working ☺
    - Reduce Row Echelon Form

data→ [2][0] dis

Challenge: *Matrix ADT*

Design an ADT 'Matrix' whose objects should be able to store a matrix of floating point values. Matrix ADT should also perform specified operations listed below related to matrices.

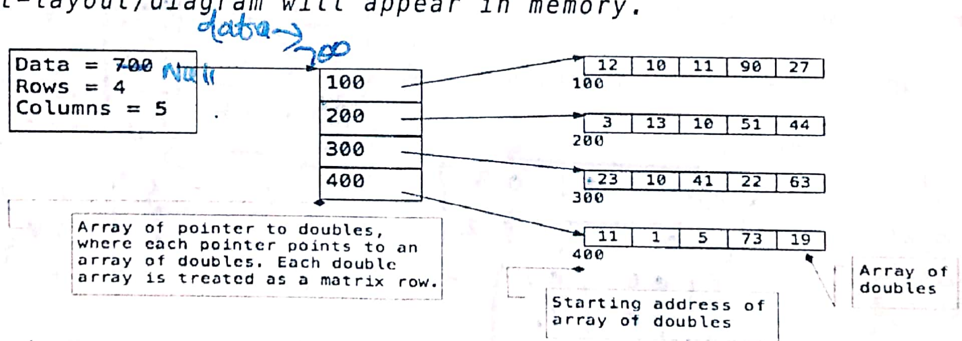
Data Members:

```
•  double  * * data;// pointer to an array of pointers whose each location
                    //points to an array of floating point values
•  int  rows;       // number of rows in matrix
•  int  columns;          // number of columns in matrix
```

Supported Operations:

1. `Matrix();` ~~(0.25)~~
   Set row and col to 0 and obviously initializes data to null as well. Represents a null matrix

2. `Matrix (int r, int c);` ~~(1.0)~~ .5
   Set the r to rows and c to columns and creates matrix structure appropriately. If user sends invalid value in r or c or both then set them to 0.
   As an example, if we pass (4,5) to constructor, then following object-layout/diagram will appear in memory.



0 3. `Matrix ( const Matrix & )` ~~(2.0)~~ 1
   The copy ctor; you know what to do. ☺

4. `~Matrix ();` ~~(1.0)~~
   Free the dynamically allocated memory.

5. `double & at(int r, int c);` ~~(0.5)~~ 0.25
   For setting or getting some value at a particular location of matrix

6. `int getRows();` ~~(0.25)~~
   returns the number of rows of the matrix.

7. `int getColumns();`
   returns the number of columns of the matrix.

~~8.~~ void display(); ~~(0.5)~~
display matrix element on console.

~~9.~~ Matrix Transpose (); (1.0)
*Find Transpose of the *this object (calling object) and returns the transpose in a new Matrix object. No change in calling object.*

~~10.~~ Matrix multiply (Matrix m2); (3.0)
Multiply this(calling object) and m2 object and return the result in a new Matrix object. No change in calling object and received object.

0

11. bool isEqual (Matrix m2); 1 ~~(1.0)~~
returns true if the calling and received objects are equal otherwise false. No change in calling object and received object.

~~12.~~ void reSize (int newRow, int newCol); 1.5 ~~(2.5)~~
resize the matrix according to new row and column. Make sure that the old matrix elements should be preserved in the new resized matrix if possible.

0

13. bool isSymmetric (); (1.0)
if $A^t = A$, then return true otherwise false. No change in calling object.

~~14.~~ void RREF (); (10.0)
it converts the matrix into reduced row echelon form.
//The First person who completes this in lab will get a reward.

Sample Run: for reduce row echelon form

```
int main()
{
    Matrix m(3,4);
    for (int i=0;i<m.getRows(); i++)
    {
        for (int j=0; j<m.getColumns(); j++)
        {
            cin>>m.at(i,j);
        }
    }
    m.RREF();
    cout<<'\n';
    return 0;
}
```

```
1       2       3       9
2       -1      1       8
3       0       -1      3

Scalar Multiply: row: 0 with: -2
   -2      -4      -6      -18
    2      -1       1        8
    3       0      -1        3

Add row: 0 --> 1
   -2      -4      -6      -18
    0      -5      -5      -10
    3       0      -1        3

Scalar Multiply: row: 0 with: 1/-2
    1       2       3        9
    0      -5      -5      -10
    3       0      -1        3
```

```
Scalar Multiply: row: 1 with: 6
    1       0       1        5
    0       6       6       12
    0      -6     -10      -24

Add row: 1 --> 2
    1       0       1        5
    0       6       6       12
    0       0      -4      -12

Scalar Multiply: row: 1 with: 1/6
    1       0       1        5
    0       1       1        2
    0       0      -4      -12

Scalar Multiply: row: 2 with: 1/-4
    1       0       1        5
```

```
Scalar Multiply: row: 0 with: -3
    -3    -6    -9   -27
     0    -5    -5   -10
     3     0    -1     3

Add row: 0 --> 2
    -3    -6    -9   -27
     0    -5    -5   -10
     0    -6   -10   -24

Scalar Multiply: row: 0 with: 1/-3
     1     2     3     9
     0    -5    -5   -10
     0    -6   -10   -24

Scalar Multiply: row: 1 with: 1/-5
     1     2     3     9
     0     1     1     2
     0    -6   -10   -24

Scalar Multiply: row: 1 with: -2
     1     2     3     9
     0    -2    -2    -4
     0    -6   -10   -24

Add row: 1 --> 0
     1     0     1     5
     0    -2    -2    -4
     0    -6   -10   -24

Scalar Multiply: row: 1 with: 1/-2
     1     0     1     5
     0     1     1     2
     0    -6   -10   -24
     0     1     1     2
     0     0     1     3

Scalar Multiply: row: 2 with: -1
     1     0     1     5
     0     1     1     2
     0     0    -1    -3

Add row: 2 --> 0
     1     0     0     2
     0     1     1     2
     0     0    -1    -3

Scalar Multiply: row: 2 with: 1/-1
     1     0     0     2
     0     1     1     2
     0     0     1     3

Scalar Multiply: row: 2 with: -1
     1     0     0     2
     0     1     1     2
     0     0    -1    -3

Add row: 2 --> 1
     1     0     0     2
     0     1     0    -1
     0     0    -1    -3

Scalar Multiply: row: 2 with: 1/-1
     1     0     0     2
     0     1     0    -1
     0     0     1     3

Program ended with exit code: 0
```

Some Examples: you may use for testing:

	Input Matrix				RREF			
1	2 -3 -2	1 -1 1	-1 2 2	8 -11 -3	1 0 0	0 1 0	0 0 1	2 3 -1
2	1 0	3 1	-1 7		1 0	0 1	-22 7	
3	1 2 1	2 2 0	1 2 1		1 0 0	0 1 0	1 0 0	

Persistence and resilience only come from having been given the chance to work through difficult problems.

-- Gever Tulley --

The Gauss-Jordan algorithm

The input of the algorithm is an $m \times n$ matrix (not necessarily square!), which is typically an augmented matrix of a linear system, however the algorithm works for *any* matrix with numerical entries.

Start with $i = 1$, $j = 1$.

1. If $a_{ij} = 0$ swap the i-th row with some other row below to guarantee that $a_{ij} \neq 0$. The non-zero entry in the (i, j)-position is called *a pivot.* If all entries in the column are zero, increase j by 1.
2. Divide the i-th row by a_{ij} to make the pivot entry $= 1$.
3. Eliminate *all* other entries in the j-th column by subtracting suitable multiples of the i-th row from the other rows.
4. Increase i by 1 and j by 1 to choose the new pivot element. Return to Step 1.

The algorithm stops after we process the last row or the last column of the matrix.

The output of the Gauss-Jordan algorithm is the matrix in *reduced row-echelon form.*

Reduced row-echelon form

A matrix is in reduced row-echelon form (RREF) if it satisfies all of the following conditions.

1. If a row has nonzero entries, then the first non-zero entry is 1 called the leading 1 in this row.
2. If a column contains a leading one then all other entries in that column are zero.
3. If a row contains a leading one the each row above contains a leading one further to the left.

The last point implies that if a matrix in rref has any zero rows they must appear as the last rows of the matrix.

© Fareed Ul Hassan Baig