Ex No: 2 Implementation of Matrix ADT

Matrix is a two dimensional array *(rows, columns)* to which various operations can be performed. The various operations are:

- Transpose of a matrix(A)
- Addition of two matrices(C ,A, B) (where A and B are of size n x n or m x n)
- Subtraction of two matrices (C, A,B) (where A and B are of size n x n or m x n)
- Multiplication of two matrices (C, A, B)
- Row operations like row switching, row multiplication by a constant, row addition,
- Upper/Lower triangular matrix (A)
- Trace and Normal of a matrix (A)
- Check Identity Matrix (A)
- Check Sparse Matrix (A)
- Check Equal Matrices (A, B)
- Inverse of a matrix (A)
- Solving linear equations

Aim: To find the transpose of a given matrix using array

Procedure:

1. Start
2. Read number of rows (r) and columns (c)
3. Declare the matrix of size m[r][c]
4. Read values of matrix cells
5. print the given matrix using loops
6. loop i=0 to r
    loop j=0 to c
        print m[j][i]
    print newline
7. End

Sample Input:			**Sample Output:**		
1	2	3	1	4	7
4	5	6	2	5	8
7	8	9	3	6	9

Result:

Thus the transpose of a matrix is done using array

The library functions malloc() is used to allocate memory from heap and free() is used for releasing the memory back to memory manager (i.e. Operating Systems).

Aim: To build matrix using dynamic memory allocation (or pointers)

Procedure:

1. Start
2. Read number of rows (r) and columns (c)
3. Allocate r*c bytes of memory using malloc() function
4. Read values of matrix cells (as shown in sample code)
5. print the given matrix using loops
6. loop i=0 to r
   loop j=0 to c
   print m[j][i]
   print newline
7. Free allocated memory using free() function
8. End

Sample Code for building Matrix using pointers (Method 1)

```
#include <stdio.h>
#include <stdlib.h>          // prototype of malloc() is given here
int main()
{
int r = 3, c = 4;
int *arr = (int *)malloc(r * c * sizeof(int));       // total 12*4 = 48 bytes are allocated
// malloc() returns void pointer. It needs to be typecast to required data type.
int i, j, count = 0;
for (i = 0; i <  r; i++)
   for (j = 0; j < c; j++)
      *(arr + i*c + j) = ++count;
// when i=0, j=0, it gets expanded to *(arr + 0 + 0) => *(arr). It is assigned the value 1
for (i = 0; i <  r; i++)
{
  for (j = 0; j < c; j++)
       printf("%d ", *(arr + i *c + j));
  printf("\n");
}
  /* Code for further processing and free dynamically allocated memory */
if(arr) free(arr);
return 0;
}
```

Sample code using pointers (Method 2)

```
int main()
{
int r = 3, c = 4, i, j, count;

int **arr = (int **)malloc(r * sizeof(int *));
for (i=0; i<r; i++)
arr[i] = (int *)malloc(c * sizeof(int));

   // Note that arr[i][j] is same as *(*(arr+i)+j)
count = 0;
for (i = 0; i <  r; i++)
for (j = 0; j < c; j++)
arr[i][j] = ++count;  // OR *(*(arr+i)+j) = ++count
}
```

Assessment Rubrics for Ex 2:

Parameters	**Max Marks**	**Actual Marks**
Uniqueness of the problem selection	10	
Code (2D Array, Pointers)	10	
Solutions for atleast 4 different problems	10	
Viva (Online Test)	10	
Adherence to the template for documentation (Record)	10	
Total	50	
Staff Signature		

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int a[3][3] = {1,2,3,4,5,6,7,8,9};
   int *p = a;   // a[0] removes warning
   printf("%d\t",a[0][0]);
   printf("%d\n",*p);
   printf("%d\t",a[0][1]);
   printf("%d\n",*(p+4));
   printf("%d\n",++(*p));
   return 0;
```

```
}
```