```c
/**
 * @author  Aviruk Basak, CSE214047, Sem 3, Year 2
 * @topic   Matrix data structure implemented in C using a 1D matrix
 * @date    29-7-2022
 * @cc      gcc -Wall -D MTX_TYPE=int -D TYPE_FORMAT='"%d"' -o dsa-int-Matrix dsa-
matrix.c
 */

# include <stdio.h>
# include <stdlib.h>

# if !defined(MTX_TYPE) || !defined(TYPE_FORMAT)
#    undef  MTX_TYPE
#    undef  TYPE_FORMAT
#    define MTX_TYPE     int
#    define TYPE_FORMAT  "%d"
# endif

# define ERR_NULLPTR       (200)
# define ERR_OUTOFBOUNDS   (201)
# define ERR_MATINCOM      (202)
# define ERR_MENUDRIVELIM  (204)

# define MENUDRIVE_LIMIT  ((size_t) 10000)

typedef MTX_TYPE  MatrixType;

typedef struct Matrix {
    MatrixType *ptr;
    size_t rows;
    size_t cols;
} *Matrix;

void mtx_nullPtrCheck(char *fname, Matrix mtx);
Matrix new_matrix(size_t rows, size_t cols);
Matrix mtx_arrayToMatrix(size_t rows, size_t cols, MatrixType arr[rows][cols]);
Matrix mtx_copy(Matrix mtx);
void mtx_print(Matrix mtx);
MatrixType mtx_get(Matrix mtx, size_t row, size_t col);
void mtx_set(Matrix mtx, size_t row, size_t col, MatrixType val);
Matrix mtx_add(Matrix mtx1, Matrix mtx2);
Matrix mtx_scale(Matrix mtx, MatrixType scalar);
Matrix mtx_multiply(Matrix mtx1, Matrix mtx2);
Matrix mtx_transpose(Matrix mtx);
void mtx_free(Matrix *mtx_ptr);

void mtx_nullPtrCheck(char *fname, Matrix mtx)
{
    if (!mtx || !mtx->ptr) {
        printf("matrix: %s: null pointer\n", fname);
        exit(ERR_NULLPTR);
    }
}

Matrix new_matrix(size_t rows, size_t cols)
{
    Matrix mtx = malloc(sizeof(struct Matrix));
    if (!mtx) {
        printf("matrix: new1: null pointer\n");
        exit(ERR_NULLPTR);
    }
    mtx->ptr = malloc(rows * cols * sizeof(MatrixType));
    mtx->rows = rows;
    mtx->cols = cols;
    mtx_nullPtrCheck("new2", mtx);
    return mtx;
}

Matrix mtx_arrayToMatrix(size_t rows, size_t cols, MatrixType arr[rows][cols])
{
    size_t i, j;
    Matrix mtx = new_matrix(rows, cols);
    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            mtx_set(mtx, i, j, arr[i][j]);
    return mtx;
}

Matrix mtx_copy(Matrix mtx)
{
    mtx_nullPtrCheck("copy", mtx);
    size_t i, j;
    Matrix cmtx = new_matrix(mtx->rows, mtx->cols);
    for (i = 0; i < cmtx->rows; i++)
        for (j = 0; j < cmtx->cols; j++)
            mtx_set(cmtx, i, j, mtx_get(mtx, i, j));
    return cmtx;
}

void mtx_print(Matrix mtx)
{
    mtx_nullPtrCheck("print", mtx);
    size_t i, j;
    for (i = 0; i < mtx->rows; i++) {
        for (j = 0; j < mtx->cols; j++)
            printf(TYPE_FORMAT "\t", mtx_get(mtx, i, j));
        printf("\n");
    }
}

MatrixType mtx_get(Matrix mtx, size_t row, size_t col)
{
    mtx_nullPtrCheck("get", mtx);
    if (row >= mtx->rows || col >= mtx->cols) {
        printf("matrix: get: index out of bounds: %zu, %zu\n", row, col);
        exit(ERR_OUTOFBOUNDS);
    }
    return mtx->ptr[row * mtx->rows + col];
}

void mtx_set(Matrix mtx, size_t row, size_t col, MatrixType val)
{
    mtx_nullPtrCheck("set", mtx);
    if (row >= mtx->rows || col >= mtx->cols) {
        printf("matrix: set: index out of bounds: %zu, %zu\n", row, col);
        exit(ERR_OUTOFBOUNDS);
    }
    mtx->ptr[row * mtx->rows + col] = val;
```

```
}

Matrix mtx_add(Matrix mtx1, Matrix mtx2)
{
    mtx_nullPtrCheck("add", mtx1);
    mtx_nullPtrCheck("add", mtx2);
    if (mtx1->rows != mtx2->rows || mtx1->cols != mtx2->cols) {
        printf("matrix: add: incompatible matrices\n");
        exit(ERR_MATINCOM);
    }
    size_t i, j;
    Matrix rmtx = new_matrix(mtx1->rows, mtx1->cols);
    for (i = 0; i < rmtx->rows; i++) {
        for (j = 0; j < rmtx->cols; j++) {
            MatrixType sum = mtx_get(mtx1, i, j) + mtx_get(mtx2, i, j);
            mtx_set(rmtx, i, j, sum);
        }
    }
    return rmtx;
}

Matrix mtx_scale(Matrix mtx, MatrixType scalar)
{
    mtx_nullPtrCheck("scale", mtx);
    size_t i, j;
    Matrix rmtx = new_matrix(mtx->rows, mtx->cols);
    for (i = 0; i < rmtx->rows; i++) {
        for (j = 0; j < rmtx->cols; j++) {
            MatrixType prod = mtx_get(mtx, i, j) * scalar;
            mtx_set(rmtx, i, j, prod);
        }
    }
    return rmtx;
}

Matrix mtx_multiply(Matrix mtx1, Matrix mtx2)
{
    mtx_nullPtrCheck("multiply", mtx1);
    mtx_nullPtrCheck("multiply", mtx2);
    if (mtx1->cols != mtx2->rows) {
        printf("matrix: multiply: incompatible matrices\n");
        exit(ERR_MATINCOM);
    }
    size_t i, j, k;
    Matrix rmtx = new_matrix(mtx1->rows, mtx2->cols);
    for (i = 0; i < mtx1->rows; i++) {
        for (j = 0; j < mtx2->cols; j++) {
            MatrixType sum = 0;
            for (k = 0; k < mtx1->cols; k++)
                sum += mtx_get(mtx1, i, k) * mtx_get(mtx2, k, j);
            mtx_set(rmtx, i, j, sum);
        }
    }
    return rmtx;
}

Matrix mtx_transpose(Matrix mtx)
{
    mtx_nullPtrCheck("transpose", mtx);
    Matrix tmtx = new_matrix(mtx->cols, mtx->rows);
    size_t i, j;
    for (i = 0; i < tmtx->rows; i++)
        for (j = 0; j < tmtx->cols; j++)
            mtx_set(tmtx, i, j, mtx_get(mtx, j, i));
    return tmtx;
}

void mtx_free(Matrix* mtx_ptr)
{
    if (mtx_ptr && *mtx_ptr && (*mtx_ptr)->ptr) {
        free((*mtx_ptr)->ptr);
        free(*mtx_ptr);
        *mtx_ptr = NULL;
    }
}

int main()
{
    int choice;
    size_t i, j, rows, cols;
    printf("enter matrix rows and cols: ");
    scanf("%zu", &rows);
    scanf("%zu", &cols);
    Matrix mtx = new_matrix(rows, cols);
    for (i = 0; i < mtx->rows; i++) {
        printf("row %zu = ", i);
        for (j = 0; j < mtx->cols; j++) {
            MatrixType val;
            scanf(TYPE_FORMAT, &val);
            mtx_set(mtx, i, j, val);
        }
    }
    do {
        printf(
            "\nchoices:\n"
            "   0: exit\n"
            "   1: print matrix\n"
            "   2: read an index\n"
            "   3: modify an index\n"
            "   4: add 2 matrices\n"
            "   5: multiply 2 matrices\n"
            "   6: compute transpose\n"
            "enter your choice: "
        );
        scanf("%d", &choice);
        printf("\n");
        switch (choice) {
            // exit
            case 0: {
                break;
            }
            // print
            case 1: {
                mtx_print(mtx);
                printf("size = %zu x %zu\n", mtx-> rows, mtx->cols);
                break;
            }
            // read
            case 2: {
                size_t row, col;
                printf("enter row and col: ");
                scanf("%zu", &row);
                scanf("%zu", &col);
                printf("mtx[%zu][%zu] = " TYPE_FORMAT "\n", row, col, mtx_get(mtx,
row, col));
                break;
            }
            // modify
            case 3: {
                size_t row, col;
                printf("enter row and col: ");
                scanf("%zu", &row);
                scanf("%zu", &col);
                MatrixType val;
                printf("enter value = ");
                scanf(TYPE_FORMAT, &val);
                mtx_set(mtx, row, col, val);
                break;
            }
            // add
            case 4: {
                printf("enter new matrix rows and cols: ");
                scanf("%zu", &rows);
                scanf("%zu", &cols);
                Matrix mtx2 = new_matrix(rows, cols);
                for (i = 0; i < mtx2->rows; i++) {
                    printf("new matrix row %zu = ", i);
                    for (j = 0; j < mtx2->cols; j++) {
                        MatrixType val;
                        scanf(TYPE_FORMAT, &val);
                        mtx_set(mtx2, i, j, val);
                    }
                }
                Matrix rmtx = mtx_add(mtx, mtx2);
                mtx_print(rmtx);
                mtx_free(&rmtx);
                mtx_free(&mtx2);
                break;
            }
            // multiply
            case 5: {
                printf("enter new matrix rows and cols: ");
                scanf("%zu", &rows);
                scanf("%zu", &cols);
                Matrix mtx2 = new_matrix(rows, cols);
                for (i = 0; i < mtx2->rows; i++) {
                    printf("new matrix row %zu = ", i);
                    for (j = 0; j < mtx2->cols; j++) {
                        MatrixType val;
                        scanf(TYPE_FORMAT, &val);
                        mtx_set(mtx2, i, j, val);
                    }
                }
                Matrix rmtx = mtx_multiply(mtx, mtx2);
                mtx_print(rmtx);
                mtx_free(&rmtx);
                mtx_free(&mtx2);
                break;
            }
            // transpose
            case 6: {
                Matrix tmtx = mtx_transpose(mtx);
                mtx_print(tmtx);
                mtx_free(&tmtx);
                break;
            }
            default: {
                printf("choice invalid\n");
            }
        }
    } while (choice);
    mtx_free(&mtx);
    return 0;
}

/* OUTPUT:

run: dsa-matrix.c
enter matrix rows and cols: 3 3
row 0 = 1 2 3
row 1 = 4 5 6
row 2 = 7 8 9

choices:
   0: exit
   1: print matrix
   2: read an index
   3: modify an index
   4: add 2 matrices
   5: multiply 2 matrices
   6: compute transpose
enter your choice: 4

enter new matrix rows and cols: 3 3
new matrix row 0 = 2 2 2
new matrix row 1 = 3 3 3
new matrix row 2 = 4 4 4
3     4     5
7     8     9
11    12    13

choices:
   0: exit
   1: print matrix
   2: read an index
   3: modify an index
   4: add 2 matrices
   5: multiply 2 matrices
   6: compute transpose
enter your choice: 1

1     2     3
4     5     6
7     8     9
size = 3 x 3
```

```
choices:
   0: exit
   1: print matrix
   2: read an index
   3: modify an index
   4: add 2 matrices
   5: multiply 2 matrices
   6: compute transpose
enter your choice: 6

1     4     7
2     5     8
3     6     9

choices:
   0: exit
   1: print matrix
   2: read an index
   3: modify an index
   4: add 2 matrices
   5: multiply 2 matrices
   6: compute transpose
enter your choice: 5

enter new matrix rows and cols: 3 2
new matrix row 0 = 1 2
new matrix row 1 = 3 4
new matrix row 2 = 5 6
22    28
49    64
76    100

choices:
   0: exit
   1: print matrix
   2: read an index
   3: modify an index
   4: add 2 matrices
   5: multiply 2 matrices
   6: compute transpose
enter your choice: 0

*/
```