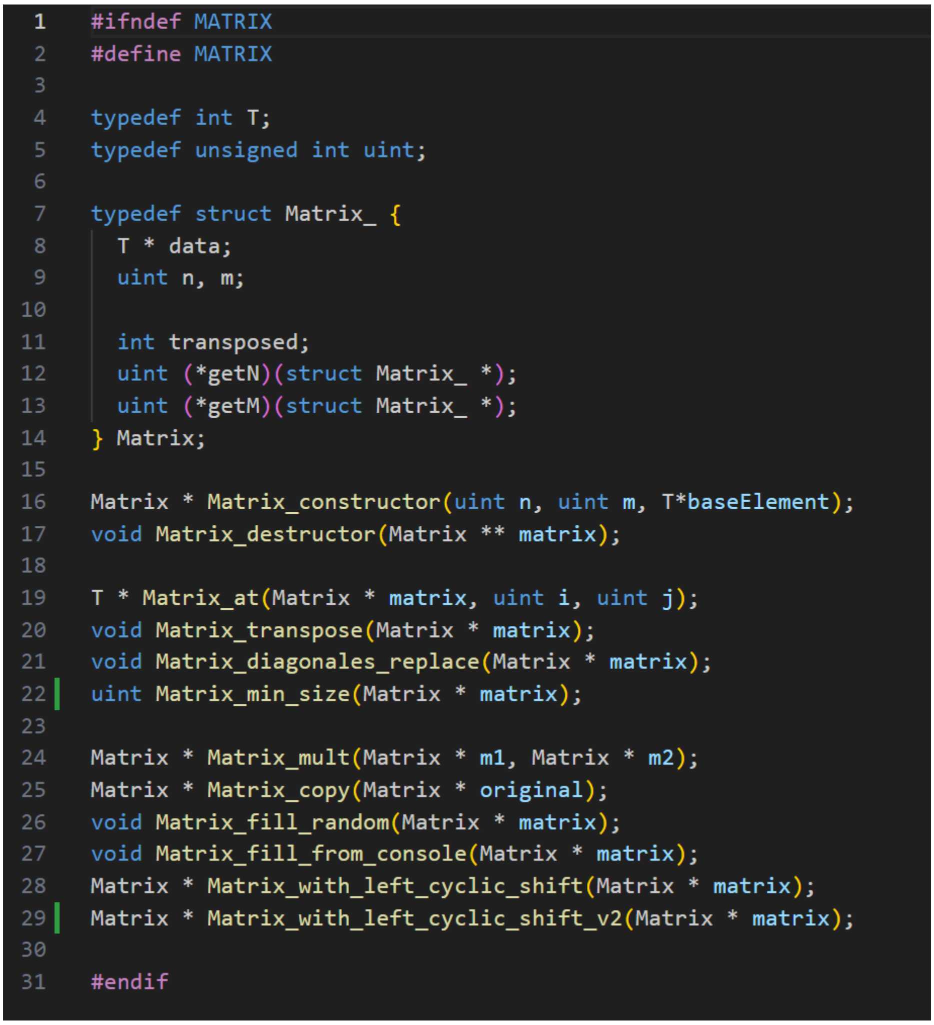
Приложение 2. Matrix.h



```
#ifndef MATRIX
#define MATRIX

typedef int T;
typedef unsigned int uint;

typedef struct Matrix_ {
  T * data;
  uint n, m;

  int transposed;
  uint (*getN)(struct Matrix_ *);
  uint (*getM)(struct Matrix_ *);
} Matrix;

Matrix * Matrix_constructor(uint n, uint m, T*baseElement);
void Matrix_destructor(Matrix ** matrix);

T * Matrix_at(Matrix * matrix, uint i, uint j);
void Matrix_transpose(Matrix * matrix);
void Matrix_diagonales_replace(Matrix * matrix);
uint Matrix_min_size(Matrix * matrix);

Matrix * Matrix_mult(Matrix * m1, Matrix * m2);
Matrix * Matrix_copy(Matrix * original);
void Matrix_fill_random(Matrix * matrix);
void Matrix_fill_from_console(Matrix * matrix);
Matrix * Matrix_with_left_cyclic_shift(Matrix * matrix);
Matrix * Matrix_with_left_cyclic_shift_v2(Matrix * matrix);

#endif
```

Приложение 2. Matrix.c

```
#include "Matrix.h"
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

uint getN(Matrix * matrix) {
  if(matrix->transposed)
    return matrix->m;
  return matrix->n;
}
uint getM(Matrix * matrix) {
  if(matrix->transposed)
    return matrix->n;
  return matrix->m;
}

uint Matrix_min_size(Matrix * matrix) {
  uint result;
  if (matrix->getN(matrix) > matrix->getM(matrix)){
    result = matrix->getM(matrix);
  } else {
    result = matrix->getN(matrix);
  }
  return result;
}

Matrix * Matrix_constructor(uint n, uint m, T*baseElement) {
  Matrix * matrix = malloc(sizeof(Matrix));
  matrix->n = n;
  matrix->m = m;

  matrix->data = malloc(sizeof(T)*n*m);
  matrix->transposed = 0;
  matrix->getM = getM;
  matrix->getN = getN;

  // Заполнение
  if(baseElement != NULL) {
    for(uint i = 0; i < n*m; ++i) {
        matrix->data[i] = *baseElement;
    }
  }
  return matrix;
}

void Matrix_destructor(Matrix ** matrix) {
  free((*matrix)->data);
  free(*matrix);
  *matrix = NULL;
}

T * Matrix_at(Matrix * matrix, uint i, uint j) {
  if(i >= matrix->getN(matrix) || j >= matrix->getM(matrix))
    return NULL;
  if(matrix->transposed)
    return &matrix->data[j * matrix->m + i];
  return &matrix->data[i * matrix->m + j];
}

void Matrix_transpose(Matrix * matrix) {
  matrix->transposed = !matrix->transposed;
}

Matrix * Matrix_mult(Matrix * m1, Matrix * m2) {
  //assert(("Invalid matrixes shape", m1->getM(m1) == m2->getN(m2)));

  int base = 0;
  Matrix * result = Matrix_constructor(m1->getN(m1), m2->getM(m2), &base);

  for(uint i = 0; i < m1->getN(m1); ++i) {
    for(uint j = 0; j < m2->getM(m2); ++j) {
      for(uint k = 0; k < m2->getN(m2); ++k) {
        *Matrix_at(result, i, j) += (*Matrix_at(m1, i, k)) * (*Matrix_at(m2, k, j));
      }
    }
  }
  return result;
}

Matrix * Matrix_copy(Matrix * original) {
  Matrix * copy = Matrix_constructor(original->getN(original), original->getM(original), NULL);
  for (uint i = 0; i < original->getN(original); ++i) {
    for (uint j = 0; j < original->getM(original); ++j) {
      *Matrix_at(copy, i, j) = *Matrix_at((Matrix *)original, i, j);
    }
  }
  return copy;
}

void Matrix_fill_random(Matrix * matrix) {
  srand(time(NULL));
  int value = 0;
  for(uint i = 0; i < matrix->m * matrix->n; ++i) {
     matrix->data[i] = rand() % 100;
  }
}

void Matrix_fill_from_console(Matrix * matrix) {
  printf("\e[2;33mPut matrix values:\e[0;0m\n");
  int value = 0;
  for(uint i = 0; i < matrix->getN(matrix); ++i) {
    for (uint j = 0; j < matrix->getM(matrix); ++j) {
      scanf("%d", Matrix_at(matrix, i, j));
    }
  }
}

void Matrix_diagonales_replace(Matrix * matrix) {
  if (matrix->m != matrix->n) {
    return;
  }
  for(uint i = 0; i < matrix->getN(matrix); ++i) {
    T * first_element = Matrix_at(matrix, i, i);
    T * second_elemet = Matrix_at(matrix, i, matrix->getN(matrix) - i - 1);
    T temp = * first_element;
    *first_element = *second_elemet;
    *second_elemet = temp;
  }
}

Matrix * Matrix_with_left_cyclic_shift(Matrix * matrix) {
  Matrix * copy = Matrix_copy(matrix);
  uint lines_count = matrix->getN(matrix);
  uint columns_count = matrix->getM(matrix);
  uint iters;
  uint line;
  uint column;

  if (lines_count > columns_count) {
    iters = columns_count;
  } else {
    iters = lines_count;
  }

  for (uint step = 0; step < iters; ++step){
    for (line = step; line < (lines_count - step); ++line){
      for (column = step; column < (columns_count - step); ++column){
        if (line == step && column != (columns_count - step - 1)) {
          *Matrix_at(copy, line, column) = *Matrix_at(matrix, line, column + 1);
        } else if (line != (lines_count - step - 1) && column == (columns_count - step - 1)) {
          *Matrix_at(copy, line, column) = *Matrix_at(matrix, line + 1, column);
        } else if (line == (lines_count - step - 1) && column != step) {
          *Matrix_at(copy, line, column) = *Matrix_at(matrix, line, column - 1);
        } else if (line != step && column == step) {
          *Matrix_at(copy, line, column) = *Matrix_at(matrix, line - 1, column);
        }
      }
    }
  }
  return copy;
}

Matrix * Matrix_with_left_cyclic_shift_v2(Matrix * matrix) {
  uint iters = Matrix_min_size(matrix) / 2;
  uint lines_count = matrix->getN(matrix);
  uint columns_count = matrix->getM(matrix);
  T base_el = 0;
  Matrix * copy = Matrix_constructor(lines_count, columns_count, &base_el);
  uint line;
  uint column;

  for (uint step = 0; step < iters; ++step){
    for (column = step; column < columns_count - 1 - step; ++column){
      *Matrix_at(copy, step, column) = *Matrix_at(matrix, step, column + 1);
      *Matrix_at(copy, lines_count - 1 - step, columns_count - 1 - column) = *Matrix_at(matrix, lines_count - 1 - step, columns_count -
1 - column - 1);
    }
    for (line = step; line < lines_count - 1 - step; ++line){
      *Matrix_at(copy, line + 1, step) = *Matrix_at(matrix, line, step);
      *Matrix_at(copy, line, columns_count - 1 - step) = *Matrix_at(matrix, line + 1, columns_count - 1 - step);
    }
  }

  if ((lines_count > columns_count) && (columns_count % 2)){
    for (line = columns_count / 2; line < lines_count - columns_count / 2; ++line){
      *Matrix_at(copy, line, columns_count / 2) = *Matrix_at(matrix, line, columns_count / 2);
    }
  } else if ((lines_count < columns_count) && (lines_count % 2)){
    for (column = lines_count / 2; column < columns_count - lines_count / 2; ++column){
      *Matrix_at(copy, lines_count / 2, column) = *Matrix_at(matrix, lines_count / 2, column);
    }
  } else if ((lines_count == columns_count) && (lines_count % 2)){
    *Matrix_at(copy, lines_count / 2, columns_count / 2) = *Matrix_at(matrix, lines_count / 2, columns_count / 2);
  }
  return copy;
}
```