Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра вычислительных систем

ОТЧЕТ
по практической работе 1

по дисциплине «**Программирование**»

Выполнил:
студент гр. ИВ-522
«__» февраля 2026 г. _____ /Родилнов П.А./

Проверил:
Ст. преподаватель кафедры ВС
«__» февраля 2026 г. _____ /Ревун А.Л./

Оценка «_____»

Новосибирск 2026

ОГЛАВЛЕНИЕ

ЗАДАНИЕ

Введение

Данная работа посвящена разработке программы для управления двумерными матрицами на языке C. Программа предоставляет возможность создавать, модифицировать и выполнять различные операции с матрицами, включая ввод данных и простые математические операции.

Задание на 3 (Сделано)

1. ***Создание структуры matrix2d**:*
    - *Реализовать структуру, включающую необходимые поля для хранения данных матрицы.*
2. ***Методы для работы с матрицами**:*
    - ***Сравнение двух матриц**:*
        - *Реализовать операторы сравнения (==, !=, >, <, >=, <=).*
    - ***Ввод/Вывод матриц**:*
        - *Реализовать методы для ввода значений матрицы с клавиатуры и вывода на экран.*
    - ***Изменение данных структуры**:*
        - *Реализовать операторы инкремента (++) и декремента (--) для матрицы.*
        - *Реализовать методы-сеттеры для изменения элементов матрицы.*
    - ***Заполнение матрицы случайными значениями**:*
        - *Реализовать метод для генерации матрицы со случайными числами в заданном диапазоне.*

Задание на 4 (Сделано)

1. ***Получение столбца/строки**:*
    - *Реализовать методы для извлечения конкретной строки или столбца из матрицы.*
2. ***Транспонирование матрицы**:*
    - *Реализовать метод, который будет возвращать транспонированную версию матрицы.*
3. ***Нахождение определителя (детерминант)**:*
    - *Реализовать метод для вычисления определителя матрицы.*

Задание на 5 (Сделано)

- ***Вычисление обратной матрицы**:*
    - *Реализовать метод для нахождения обратной матрицы, если она существует.*

Итог

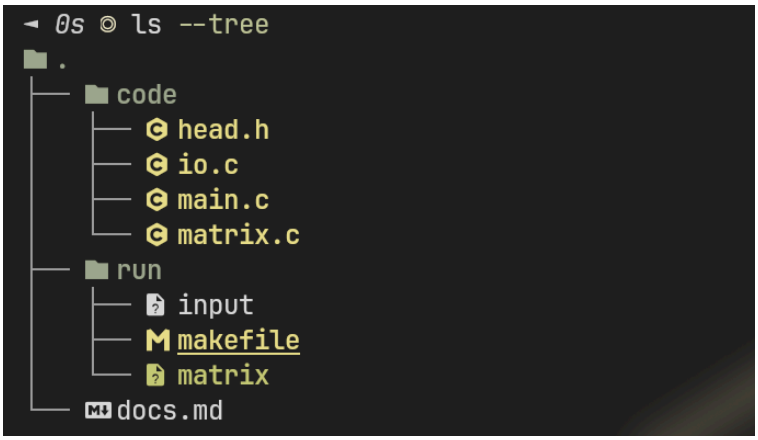
*В результате выполнения задания должна быть реализована полноценная структура **matrix2d** с необходимыми методами, обеспечивающими работу с двумерными матрицами на уровне различных операций.*

ОПИСАНИЕ РАБОТЫ

Структура проекта

Проект состоит из следующих файлов:

- *head.h: заголовочный файл, содержащий определения структур и объявление функций.*
- *io.c: файл, содержащий функции для ввода и вывода данных.*
- *matrix.c: файл, реализующий функции для работы с матрицами.*
- *main.c: основной файл, содержащий реализованный интерфейс программы.*
- *makefile: файл, управляющий процессом сборки.*



Описание структуры и функциональности

Структура MATRIX2D

Структура MATRIX2D определена в заголовочном файле head.h и содержит:

- ***rows**: количество строк матрицы.*
- ***cols**: количество столбцов матрицы.*
- ***data**: указатель на двумерный массив целых чисел.*

Основные функции

1. ***create_matrix2d**: Создает матрицу с заданными количеством строк и столбцов.*
2. ***input_matrix**: Вводит значения в матрицу.*
3. ***get_matrix**: Выводит значения матрицы.*
4. ***logic_matrix**: Выполняет логическую операцию над двумя матрицами.*
5. ***edit_matrix**: Изменяет значения в матрице.*
6. ***matrix_random**: Заполняет матрицу случайными значениями.*
7. ***copy_matrix**: Копирует одну матрицу в другую.*
8. ***free_matrix**: Освобождает память, занятую матрицей.*
9. ***transp_matrix**: Транспонирует матрицу.*
10. ***determ_matrix**: Находит определитель матрицы.*
11. ***obr_matrix**: Вычисляет обратную матрицу.*
12. ***is_Matrix_Exist**: Проверяет существование матриц*

Описание процесса работы программы

Интерфейсный ввод

Программа ожидает команд от пользователя, которые выполняют следующие операции:

1. ***create**: создание матрицы.*
    - *Ожидает номер матрицы, количество строк и столбцов.*
2. ***input**: ввод значений в матрицу.*
    - *Ожидает номер матрицы и значения.*
3. ***getmatrix**: вывод значений матрицы.*
    - *Ожидает номер матрицы.*
4. ***logic**: выполнение логической операции.*
    - *Ожидает два номера матриц и сами операции.*
5. ***random**: заполнение матрицы случайными числами.*
    - *Ожидает номер матрицы.*
6. ***copy**: копирование матрицы.*
    - *Ожидает номер матрицы-источника и номер матрицы-назначения.*
7. ***exit**: завершение программ*

Описывается ход работы над заданием с приложением снимков экрана;

ПРОЦЕСС ВЫПОЛНЕНИЯ РАБОТ

1. Инициализация структуры хранения матриц

На первом этапе была реализована возможность работы с несколькими матрицами одновременно.
Для этого в функции main() был создан массив указателей на структуры MATRIX2D.

Это позволяет динамически создавать, удалять и изменять матрицы во время работы программы.

```
1  #include "head.h"
2
3  int main(void){
4
5      MATRIX2D** __matrixAddr = (MATRIX2D**)calloc(MAX_MATRIX_ALIVE,
6              sizeof(MATRIX2D*));
7      if(__matrixAddr == NULL) {
8          printf("ERROR.__matrixAddr NotExist");return 1;}
```

После выделения памяти выполняется проверка на успешность создания массива.
В случае ошибки программа завершает работу с сообщением об ошибке.

---

2. Реализация структуры MATRIX2D

Структура MATRIX2D определена в заголовочном файле head.h и содержит:

- rows — количество строк
- cols — количество столбцов
- data — указатель на динамический двумерный массив

Динамическое выделение памяти позволяет работать с матрицами произвольного размера.

```c
// matrix.c
typedef struct {
    int rows;
    int cols;
    int** data;
} MATRIX2D;
```

---

3. Создание матрицы

Функция create_matrix (или create_matrix2d) реализует:

- выделение памяти под структуру
- выделение памяти под массив указателей строк
- выделение памяти под каждую строку
- инициализацию значений

Алгоритм создания:

1. Проверка корректности размеров
2. Выделение памяти под структуру
3. Выделение памяти под строки
4. Обработка возможных ошибок выделения

```
1  MATRIX2D* create_matrix2d(int rows,int cols) {
2     MATRIX2D* m =(MATRIX2D*)malloc(
3  sizeof(MATRIX2D));
4     m->rows = rows;
5     m->cols = cols;
6
7     m->data = (int**)malloc(rows * sizeof(int*));
8
9     for (int i = 0; i < rows; i++){
10       m->data[i] = (int*)malloc(cols * sizeof(int));
11    }
12
13    return m;
14 }
```

---

4. Копирующий конструктор

Функция copy_matrix реализует глубокое копирование:

- создаётся новая матрица того же размера
- копируются все элементы
- память не разделяется

```
1  void copy_matrix(MATRIX2D** __addr,unsigned char n1, unsigned char n2){
2      __addr[n2] = create_matrix2d(__addr[n1]->rows,__addr[n1]->cols);
3
4      for(int i = 0; i<__addr[n1]->rows;i++){
5          for(int j = 0; j<__addr[n1]->cols;j++){
6              __addr[n2]->data[i][j] = __addr[n1]->data[i][j];
7          }
8      }
9  }
10
```

---

5. Освобождение памяти

Для предотвращения утечек памяти была реализована функция free_matrix.

Алгоритм работы функции:

1. Проверка существования матрицы
2. Освобождение каждой строки
3. Освобождение массива указателей
4. Освобождение самой структуры

```
1  void free_matrix(MATRIX2D** __addr,unsigned char
2  n){
3      if (__addr[n] == NULL) return;
4      MATRIX2D* m = __addr[n];
5      for (int i = 0; i < m->rows; i++) {
6          free(m->data[i]);
7      }
8      free(m->data);
9      free(m);
10     __addr[n] = NULL;
11 }
```

---

6. Конструктор по умолчанию

В явном виде отдельной функции «по умолчанию» нет, но поведение по умолчанию реализуется через функцию:

void is_Matrix_Exist(...)

Если матрица не существует, создаётся новая размером 4×4 и заполняется случайными значениями.

```
1  void is_Matrix_Exist(MATRIX2D** __addr,unsigned char n){
2     if(n > MAX_MATRIX_ALIVE) n = MAX_MATRIX_ALIVE-1;
3     if(__addr[n] == NULL){
4        __addr[n] = create_matrix2d(4,4);
5        matrix_random(__addr,n);
6        }
7
8     return;
9  }
```

Это фактически выполняет роль конструктора по умолчанию.

---

7. Реализация интерфейса команд

Программа работает в цикле обработки команд пользователя.

Основной цикл:

- считывает команду
- определяет её тип
- вызывает соответствующую функцию

```
1   char mode;
2    do{
3       char inbuff;
4       char _buff[BUFF_MAX];
5       char n1,n2;
6       int num1,num2;
7       mode = buff_input(&inbuff,_buff);
8       {
9          switch(mode){
10            case 1:
11               n1 = get_matrix_num();
12               is_Matrix_Exist(__matrixAddr,n1);
13               input_matrix(__matrixAddr,n1);
14               break;
15
16            case 2:  // output
17               n1 = get_matrix_num();
18               is_Matrix_Exist(__matrixAddr,n1);
19               get_matrix(__matrixAddr,n1);
20
21               break;
22
23            case 3:  // logic
24               n1 = get_matrix_num();
25               n2 = get_matrix_num();
26               is_Matrix_Exist(__matrixAddr,n1);
27               is_Matrix_Exist(__matrixAddr,n2);
28               logic_matrix(__matrixAddr,n1,n2);
29               break;
30
31            case 4:  // edit
32               n1 = get_matrix_num();
33               is_Matrix_Exist(__matrixAddr,n1);
34               edit_matrix(__matrixAddr,n1);
35               break;
36
37            case 5:  // random
38               n1 = get_matrix_num();
39               is_Matrix_Exist(__matrixAddr,n1);
40               matrix_random(__matrixAddr,n1);
41               break;
42
43            case 6:  // copy
```

```
44                  n1 = get_matrix_num();
45                  n2 = get_matrix_num();
46                  is_Matrix_Exist(__matrixAddr,n1);
47                  free_matrix(__matrixAddr,n2);
48                  copy_matrix(__matrixAddr,n1,n2);
49                  break;
50
51              case 7:  // destroy
52                  n1 = get_matrix_num();
53                  free_matrix(__matrixAddr,n1);
54                  break;
55
56              case 8:  // get colum
57                  n1 = get_matrix_num();
58                  num1 = get_num();
59                  is_Matrix_Exist(__matrixAddr,n1);
60                  get_matrix_col(__matrixAddr,n1,num1);
61                  break;
62
63              case 9:  // get row
64                  n1 = get_matrix_num();
65                  num1 = get_num();
66                  is_Matrix_Exist(__matrixAddr,n1);
67                  get_matrix_row(__matrixAddr,n1,num1);
68                  break;
69
70              case 10: // transp
71                  n1 = get_matrix_num();
72                  is_Matrix_Exist(__matrixAddr,n1);
73                  transp_matrix(__matrixAddr,n1);
74                  break;
75
76              case 11: // determ
77                  n1 = get_matrix_num();
78                  is_Matrix_Exist(__matrixAddr,n1);
79                  printf("%lld",determ_matrix(__matrixAddr,n1));
80
81                  break;
82
83              case 12: //obr
84                  n1 = get_matrix_num();
85                  is_Matrix_Exist(__matrixAddr,n1);
86                  obr_matrix(__matrixAddr,n1);
```

```
87                  break;
88
89              case 13:
90                  n1 = get_matrix_num();
91                  num1 = get_num();
92                  num2 = get_num();
93                  free_matrix(__matrixAddr,n1);
94                  create_matrix(__matrixAddr,n1,num1,num2);
95                  break;
96
97              case 15:
98                  putchar('\n');
99                  putchar('\n');
100                 break;
101
102             case 0:  // wrong input
103                 break;
104
105             case -1: return 0;
106             }
107         }
108     }while(mode>=0);
109
110     return 0;
111 }
```

Такой подход реализует примитивный командный интерфейс (консольное управление).

---

8. Обработка ввода команд

Для анализа пользовательского ввода реализована функция buff_input.

Она:

- считывает строку посимвольно
- сравнивает её с набором доступных команд
- возвращает номер команды

```c
#include "head.h"
char buff_input(char *inbuff, char *_buff){
   *inbuff = 0;
   { // INPUT
      char ch;
      while((ch=getchar())>' '&&*inbuff<BUFF_MAX){
         *(_buff+*inbuff)=(int)ch;
         (*inbuff)++;
      }
      if(ch<= 4)return -1;
   }


   if (_buff[0]>='a' && _buff[0] <= 'z')
      {   // Is input - command?
      const char NUM_OF_CHAR_COMMANDS = 15;
      char * _charCommands[NUM_OF_CHAR_COMMANDS]; //
_charCommands[*]["char"]
      _charCommands[0]   = (char[]){"input"};
      _charCommands[1]   = (char[]){"getmatrix"};
      _charCommands[2]   = (char[]){"logic"};
      _charCommands[3]   = (char[]){"edit"};
      _charCommands[4]   = (char[]){"random"};
      _charCommands[5]   = (char[]){"copy"};
      _charCommands[6]   = (char[]){"destruct"};
      _charCommands[7]   = (char[]){"getcolum"};
      _charCommands[8]   = (char[]){"getrow"};
      _charCommands[9]   = (char[]){"transp"};
      _charCommands[10]   = (char[]){"determ"};
      _charCommands[11]   = (char[]){"obr"};
      _charCommands[12]   = (char[]){"create"};
      _charCommands[13]   = (char[]){"n"};
      _charCommands[14]   = (char[]){"exit"};


      char cmd =
cmd_buff(_buff,_charCommands,NUM_OF_CHAR_COMMANDS);
      if(cmd==15)return -1; //exit
      return cmd; //Вернуть комману
   }

   return 0;
}
```

Для сопоставления строки с командами используется функция cmd_buff.

```
1  char cmd_buff(char* _buff,char** _charCommands,char NUM_OF_CHAR_COMMANDS){
2     for (int i = 0;i<NUM_OF_CHAR_COMMANDS;i++){
3        char flag = 1;
4        for(int j = 0;*(_charCommands[i]+j)!='\0';j++){
5           if (_buff[j]!=*(_charCommands[i]+j)){
6              flag = 0;
7              break;
8           }
9        }
10       if(flag) return i+1;//Вернуть комманду
11    }
12    return 0;
13 }
14
```

---

9. Ввод и вывод матрицы

Функция input_matrix позволяет пользователю ввести значения элементов.

Алгоритм:

1. Проход по строкам
2. Проход по столбцам
3. Ввод значения через get_num
4. Сохранение в data[i][j]

```
1   void input_matrix(MATRIX2D** __addr,unsigned char n){
2      int** data = __addr[n]->data;
3
4      for(int rows = 0; rows < __addr[n]->rows;rows++){
5         for(int cols = 0; cols < __addr[n]->cols;cols++){
6               int num = get_num();
7               data[rows][cols] = num;
8            }
9         }
10     }
11
12     void get_matrix(MATRIX2D** __addr,unsigned char
13  n){
14        int rows = __addr[n]->rows;
15        int cols = __addr[n]->cols;
16
17     for(int r = 0;r<rows;r++){
18        for(int c = 0;c<cols;c++){
19           printf("%d ",__addr[n]->data[r][c]);
20        }
21        putchar('\n');
22     }
23  }
24
```

Функция get_matrix выводит элементы в виде таблицы.

```
1  void get_matrix_row(MATRIX2D** __addr,unsigned char n,int
2  r){
3      if(r >= __addr[n]->rows)r = __addr[n]->rows - 1;
4      if(r<0)r = 0;
5      for(int i = 0;i<__addr[n]->cols;i++){
6          printf("%d ",__addr[n]->data[r][i]);
7      }
8      putchar('\n');
9  }
```

---

10. Изменение данных структуры

Setter (редактирование элемента)

Функция edit_matrix позволяет:

- выбрать позицию (point)
- прибавить значение (add)
- вычесть значение (sub)
- либо установить новое значение

Реализована проверка выхода за пределы INT_MAX и INT_MIN.

```
1   void edit_matrix(MATRIX2D** __addr, unsigned char n) {
2       char inbuff;
3       char _buff[BUFF_MAX];
4       int** data = __addr[n]->data;
5       int r = 0, c = 0;
6       char mode = 0;
7
8       char* _locStrings[3];
9       _locStrings[0] = (char[]){"point"};
10      _locStrings[1] = (char[]){"add"};
11      _locStrings[2] = (char[]){"sub"};
12
13      while (r < __addr[n]->rows) {
14          char res = buff_input(&inbuff, _buff);
15          if (res == -1) break;
16
17          if (_buff[0] == 'n' && inbuff == 1) break;
18
19          char loc_cmd = cmd_buff(_buff, _locStrings, 3);
20
21          if (loc_cmd == 1) {
22              buff_input(&inbuff, _buff);
23              long pr = 0, pc = 0;
24              int i = 0;
25              for (; i < inbuff && (_buff[i] != ':' && _buff[i] != ',' && _buff[i] != '.'); i++)
26                  if (_buff[i] >= '0' && _buff[i] <= '9') pr = pr * 10 + (_buff[i] - '0');
27              for (i++; i < inbuff; i++)
28                  if (_buff[i] >= '0' && _buff[i] <= '9') pc = pc * 10 + (_buff[i] - '0');
29
30              if (pr < __addr[n]->rows && pc < __addr[n]->cols) {
31                  r = (int)pr;
32                  c = (int)pc;
33              }
34              continue;
35          }
36
37          if (loc_cmd == 2) { mode = 1; continue; }
38          if (loc_cmd == 3) { mode = 2; continue; }
39
40          long val = 0;
41          int i = 0, sign = 1;
42          if (inbuff > 0 && _buff[0] == '-') { sign = -1; i++; }
43          for (; i < inbuff; i++) {
```

```
44              if (_buff[i] >= '0' && _buff[i] <= '9')
45                  val = val * 10 + (_buff[i] - '0');
46          }
47          val *= sign;
48
49          if (mode == 1) {
50              long res_val = (long)data[r][c] + val;
51              data[r][c] = (res_val > INT_MAX) ? INT_MAX : (res_val < INT_MIN) ? INT_MIN :
52  (int)res_val;
53          } else if (mode == 2) {
54              long res_val = (long)data[r][c] - val;
55              data[r][c] = (res_val > INT_MAX) ? INT_MAX : (res_val < INT_MIN) ? INT_MIN :
56  (int)res_val;
57          } else {
58              data[r][c] = (val > INT_MAX) ? INT_MAX : (val < INT_MIN) ? INT_MIN : (int)val;
59          }
60
61          if (++c >= __addr[n]->cols) {
62              c = 0;
63              r++;
64          }
65      }
66  }
67
68
69
70
71
```

Инкремент и декремент

Отдельных функций ++ и -- нет, но их логика реализована через режимы add и sub внутри edit_matrix.

Это позволяет изменять элементы динамически.

11. Заполнение случайными значениями

Функция matrix_random заполняет матрицу случайными числами в заданном диапазоне.

Используется генератор псевдослучайных чисел rand().

```
1  int get_rand(int min, int max) {
2      return (double)rand() / (RAND_MAX + 1.0) * (max - min) + min;
3  }
4
5  void matrix_random(MATRIX2D** __addr,unsigned char n){
6      unsigned int lo, hi;
7      __asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
8      srand(((unsigned long long)hi << 32) | lo);
9
10     for (int r = 0; r < __addr[n]->rows; r++){
11         for(int c = 0; c < __addr[n]->cols; c++){
12             __addr[n]->data[r][c] = get_rand(-100, 100);
13         }
14     }
15 }
```

---

12. Логические операции и сравнение

Функция logic_matrix реализует операции сравнения между двумя матрицами:

== != > < >= <=

Перед выполнением проверяется:

- существование матриц

```
1  void logic_matrix(MATRIX2D** __addr,unsigned char n1,unsigned char n2){
2     char inbuff;
3     char _buff[BUFF_MAX];
4     int** data1 = __addr[n1]->data;
5     int** data2 = __addr[n2]->data;
6
7     char cmd = 0;
8     {
9        buff_input(&inbuff,_buff);
10
11       const char NUM_OF_CHAR_COMMANDS = 6;
12       char * _charCommands[NUM_OF_CHAR_COMMANDS]; // _charCommands[*]["char"]
13       _charCommands[0]    = (char[]){"=="};
14       _charCommands[1]    = (char[]){"!="};
15       _charCommands[2]    = (char[]){">="};
16       _charCommands[3]    = (char[]){"<="};
17       _charCommands[4]    = (char[]){">"};
18       _charCommands[5]    = (char[]){"<"};
19
20       cmd = cmd_buff(_buff,_charCommands,NUM_OF_CHAR_COMMANDS);
21    }
22
23    LLI cost1 = 0;
24    {
25       for(int r = 0;r<__addr[n1]->rows;r++)
26          for(int c = 0;c<__addr[n1]->cols;c++)
27             cost1 += data1[r][c];
28    }
29    LLI cost2 = 0;
30    {
31       for(int r = 0;r<__addr[n2]->rows;r++)
32          for(int c = 0;c<__addr[n2]->cols;c++)
33             cost2 += data2[r][c];
34    }
35
36    switch(cmd){
37       case 0: //Null
38             putchar('-');putchar('1');
39          break;
40       case 1: // ==
41             putchar(cost1==cost2?'1':'0');
42          break;
43       case 2: // !=
```

```
44              putchar(cost1!=cost2?'1':'0');
45              break;
46          case 3: //>=
47              putchar(cost1>=cost2?'1':'0');
48              break;
49          case 4: //<=
50              putchar(cost1<=cost2?'1':'0');
51              break;
52          case 5: // >
53              putchar(cost1>cost2?'1':'0');
54              break;
55          case 6: // <
56              putchar(cost1<cost2?'1':'0');
57              break;
58      }
59  }
```

13. Получение строки и столбца

Получение столбца

Проверяется выход за границы.
Если индекс выходит за диапазон — корректируется.

```
1  void get_matrix_col(MATRIX2D** __addr,unsigned char n,int c){
2      if(c >= __addr[n]->cols)c = __addr[n]->cols - 1;
3      if(c<0)c = 0;
4      for(int i = 0;i<__addr[n]->rows;i++){
5          printf("%d ",__addr[n]->data[i][c]);
6      }
7      putchar('\n');
8  }
9
```

Получение строки

Аналогично реализована функция получения строки.

```
1  void get_matrix_row(MATRIX2D** __addr,unsigned char n,int r){
2      if(r >= __addr[n]->rows)r = __addr[n]->rows - 1;
3      if(r<0)r = 0;
4      for(int i = 0;i<__addr[n]->cols;i++){
5          printf("%d ",__addr[n]->data[r][i]);
6      }
7      putchar('\n');
8  }
```

14. Транспонирование матрицы

Функция transp_matrix выполняет обмен строк и столбцов.

Алгоритм:

1. Создание новой матрицы размера cols x rows
2. Перенос элементов по формуле:

   new[j][i] = old[i][j]

```
1  void transp_matrix(MATRIX2D** __addr,unsigned char n){
2     MATRIX2D* tr = create_matrix2d(
3        __addr[n]->cols,__addr[n]->rows);
4
5
6     int** dataDef = __addr[n]->data;
7     int** dataRes = tr->data;
8
9     for(int r = 0;r<__addr[n]->rows;r++)
10       for(int c = 0;c<__addr[n]->cols;c++)
11          dataRes[c][r] = dataDef[r][c];
12
13    free_matrix(__addr,n);
14
15    __addr[n] = tr;
16 }
```

---

15. Нахождение определителя

Функция determ_matrix вычисляет детерминант квадратной матрицы.

Перед вычислением проверяется:

- равенство количества строк и столбцов

Реализация выполнена рекурсивным методом разложения по минору.

```
LLI determ_matrix(MATRIX2D** __addr, unsigned char n) {
    int size = __addr[n]->rows;
    int** data = __addr[n]->data;

    if (size == 1) return data[0][0];
    if (size == 2) return (LLI)data[0][0] * data[1][1] - (LLI)data[0][1] * data[1][0];

    double det = 1.0;
    double** temp = (double**)malloc(size * sizeof(double*));
    for (int i = 0; i < size; i++) {
        temp[i] = (double*)malloc(size * sizeof(double));
        for (int j = 0; j < size; j++) temp[i][j] = (double)data[i][j];
    }

    for (int i = 0; i < size; i++) {
        int pivot = i;
        for (int j = i + 1; j < size; j++) {
            double val1 = temp[j][i] < 0 ? -temp[j][i] : temp[j][i];
            double val2 = temp[pivot][i] < 0 ? -temp[pivot][i] : temp[pivot][i];
            if (val1 > val2) pivot = j;
        }

        if (pivot != i) {
            double* swap = temp[i];
            temp[i] = temp[pivot];
            temp[pivot] = swap;
            det *= -1;
        }

        double current_val = temp[i][i] < 0 ? -temp[i][i] : temp[i][i];
        if (current_val < 1e-9) {
            for (int k = 0; k < size; k++) free(temp[k]);
            free(temp);
            return 0;
        }

        det *= temp[i][i];

        for (int j = i + 1; j < size; j++) {
            double factor = temp[j][i] / temp[i][i];
            for (int k = i + 1; k < size; k++) {
                temp[j][k] -= factor * temp[i][k];
            }
```

```
44          }
45      }
46
47      LLI final_det;
48      if (det >= 0) final_det = (LLI)(det + 0.5);
49      else final_det = (LLI)(det - 0.5);
50
51      for (int i = 0; i < size; i++) free(temp[i]);
52      free(temp);
53
54      return final_det;
55  }
56
```

---

16. Нахождение обратной матрицы

Функция obr_matrix реализует вычисление обратной матрицы.

Алгоритм:

1. Вычисление определителя
2. Проверка, что он не равен нулю
3. Построение матрицы алгебраических дополнений
4. Транспонирование
5. Деление на определитель

```
1  void obr_matrix(MATRIX2D** __addr, unsigned char n) {
2      int size = __addr[n]->rows;
3      if (size != __addr[n]->cols) return;
4
5      LLI d = determ_matrix(__addr, n);
6      if (d == 0) return;
7
8      double** temp = (double**)malloc(size * sizeof(double*));
9      for (int i = 0; i < size; i++) {
10         temp[i] = (double*)malloc(2 * size * sizeof(double));
11         for (int j = 0; j < size; j++) {
12             temp[i][j] = (double)__addr[n]->data[i][j];
13             temp[i][j + size] = (i == j ? 1.0 : 0.0);
14         }
15     }
16
17     for (int i = 0; i < size; i++) {
18         int pivot = i;
19         for (int j = i + 1; j < size; j++) {
20             double v1 = temp[j][i] < 0 ? -temp[j][i] : temp[j][i];
21             double v2 = temp[pivot][i] < 0 ? -temp[pivot][i] : temp[pivot][i];
22             if (v1 > v2) pivot = j;
23         }
24
25         double* row_ptr = temp[i];
26         temp[i] = temp[pivot];
27         temp[pivot] = row_ptr;
28
29         double divisor = temp[i][i];
30         for (int j = i; j < 2 * size; j++) {
31             temp[i][j] /= divisor;
32         }
33
34         for (int k = 0; k < size; k++) {
35             if (k != i) {
36                 double factor = temp[k][i];
37                 for (int j = i; j < 2 * size; j++) {
38                     temp[k][j] -= factor * temp[i][j];
39                 }
40             }
41         }
42     }
43
```

```
44      for (int i = 0; i < size; i++) {
45          for (int j = 0; j < size; j++) {
46              double val = temp[i][j + size];
47              if (val >= 0) __addr[n]->data[i][j] = (int)(val + 0.5);
48              else __addr[n]->data[i][j] = (int)(val - 0.5);
49          }
50      }
51
52      for (int i = 0; i < size; i++) free(temp[i]);
53      free(temp);
54  }
```

Итог

В работе реализованы:

- динамическая структура MATRIX2D
- вложенное выделение памяти
- конструктор
- копирующий конструктор
- конструктор по умолчанию
- деструктор
- setter
- логические операции
- получение строки и столбца
- транспонирование
- определитель
- обратная матрица
- Все объекты структуры размещаются в динамической памяти и корректно освобождаются.