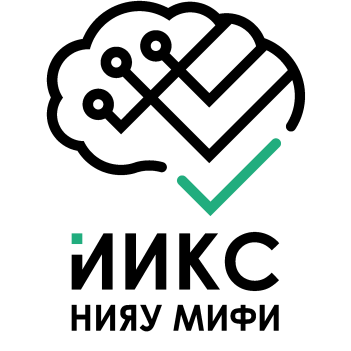
**Национальный исследовательский ядерный университет «МИФИ»**

**Институт интеллектуальных кибернетических систем**

**Кафедра №12 «Компьютерные системы и технологии»**



**ОТЧЕТ**

**О выполнении лабораторной работы №4**

**«Работа с матрицами»**

Студент: Кафанов С.П.

Группа: Б21-515

Преподаватель: Храпов А.С.

Москва — 2021

**Задача. Вариант №10**

Необходимо спроектировать и разработать на языке C программу, осуществляющую обработку матриц, представленных в виде двумерных массивов

Программа должна иметь следующую функциональность:

1. Ввод данных:

• Чтение исходных данных из стандартного потока ввода («с клавиатуры»).

• Чтение исходных данных из бинарного файла.

• Генерация случайных исходных данных.

2. Вывод данных:

• Запись исходных данных в стандартный поток вывода («на экран»).

• Запись результата в стандартный поток вывода («на экран»).

• Запись исходных данных в бинарный файл.

• Запись результата в бинарный файл.

3. Обработка данных:

• Вставка строки в матрицу по индексу.

• Вставка столбца в матрицу по индексу.

• Удаление строки из матрицы по индексу.

• Удаление столбца из матрицы по индексу.

•Найти все миноры квадратной матрицы.

**Использованные типы данных.**

*Int –* для работы с простыми целочисленными данными

*Longdouble –* для чисел в матрице

*Size\_t –* для сортировок: арифметика указателя void, хранение неопределённо большого числа

*Char –* для строк, чисел небольшого диапазона и работы с байтами памяти

*\* -* для работы с динамической памятью и передачи переменных в функцию, просто раоты с указателями

*Void \* -* для написания универсальных сортировок

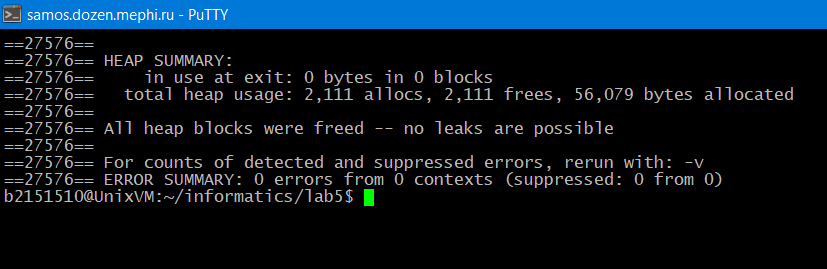
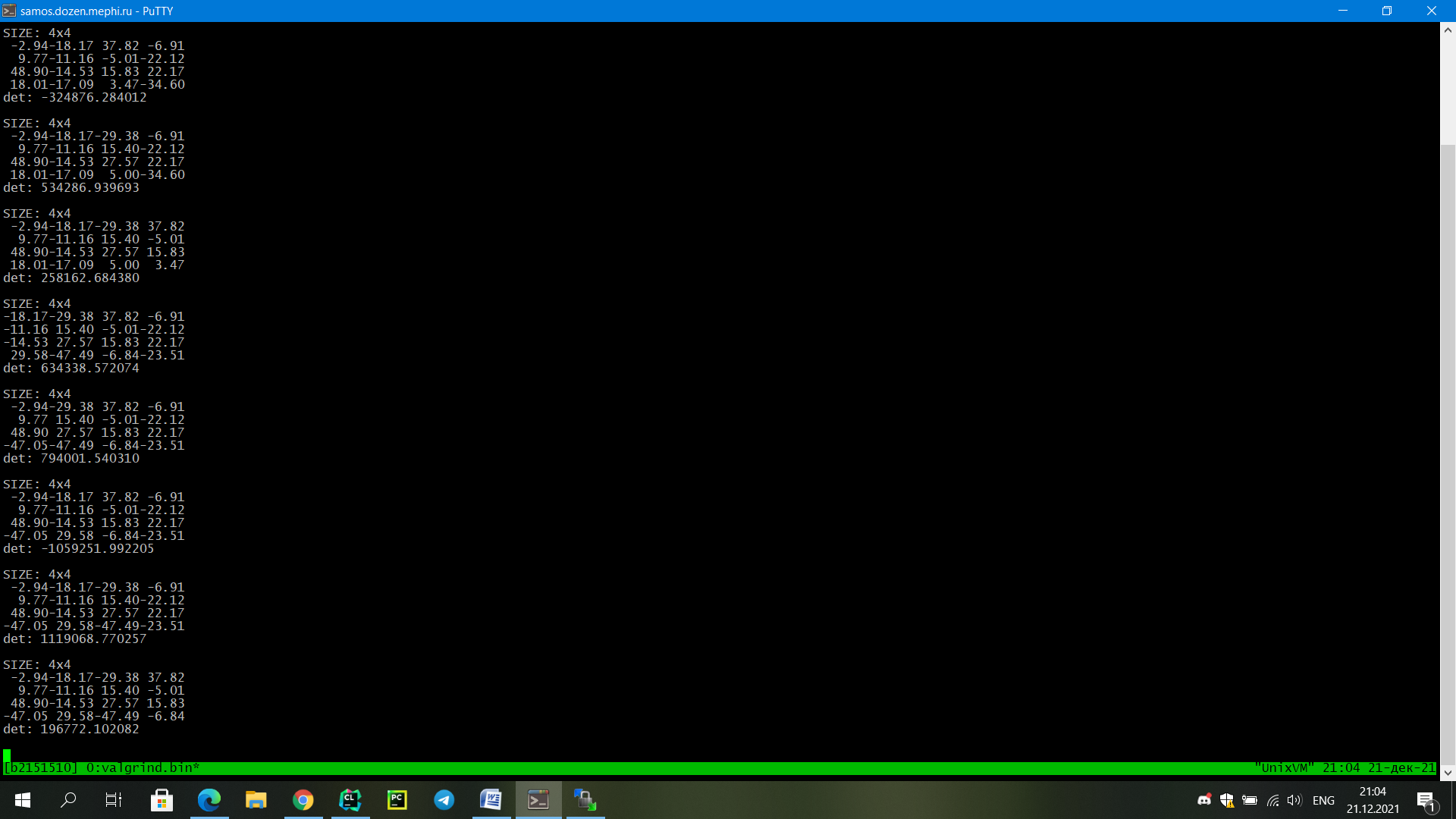
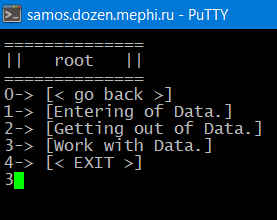
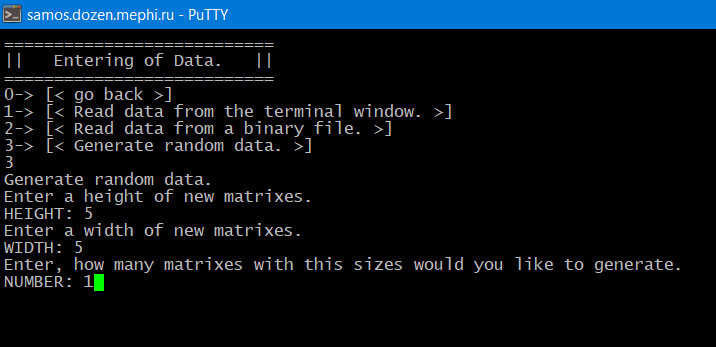
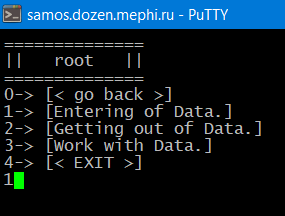
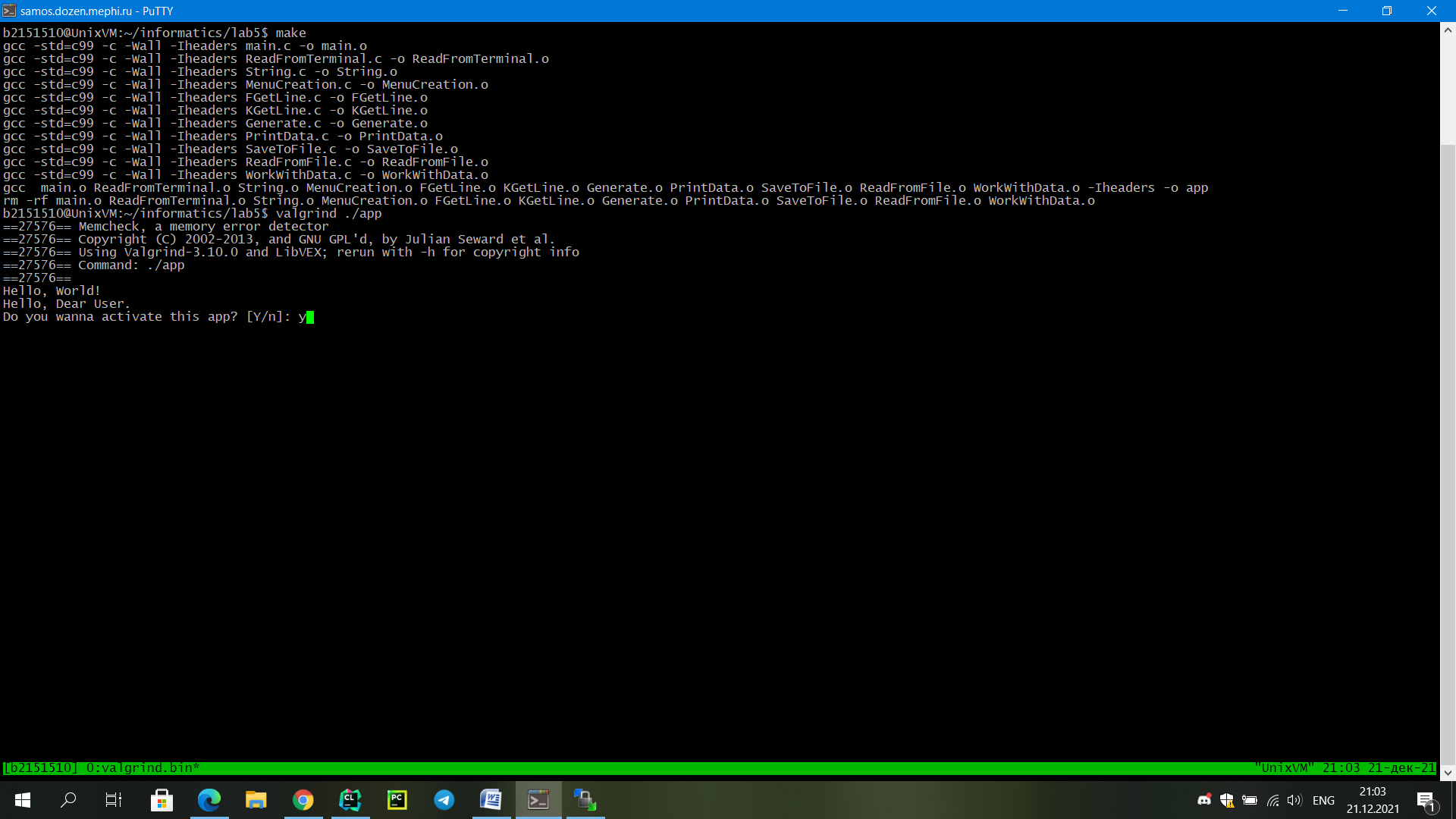
**Код.**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include "MatrixStruct.h"  
#include "KGetLine.h"  
#include "String.h"  
  
void printMatrix(Matrix\* m) {  
 printf("SIZE: %ldx%ld\n", m->height, m->width);  
 for (size\_t ii = 0; ii < m->height; ++ii) {  
 for (size\_t k = 0; k < m->width; ++k)  
 printf("%6.2lf", m->ptr[ii \* m->width + k]);  
 printf("\n");  
 }  
}  
  
Matrix \* insert\_row(Matrix\* MATRIX, double\* m,int i) {  
 size\_t w = MATRIX->width, h = MATRIX->height;  
 Matrix \* M = malloc(sizeof(Matrix));  
 M->ptr = malloc(sizeof(double)\*(h+1)\*w);  
 M->width = w;  
 M->height = h+1;  
 h++;  
  
 double \* donor = MATRIX->ptr;  
 double \* recipient = M->ptr;  
 if (i != 0)  
 memcpy(recipient, donor, w\*i\*sizeof(double));  
 donor = m;  
 recipient = M->ptr + w\*i;  
 memcpy(recipient, donor, w\*sizeof(double));  
 donor = MATRIX->ptr + w\*i;  
 recipient = M->ptr + w\*(i+1);  
 memcpy(recipient, donor, ((h-1)\*w - w\*i)\*sizeof(double));  
 printMatrix(M);  
 return M;  
}  
  
Matrix \* insert\_column(Matrix\* MATRIX, double\* m,int i) {  
 size\_t w = MATRIX->width, h = MATRIX->height;  
 Matrix \* M = malloc(sizeof(Matrix));  
 M->ptr = malloc(sizeof(double)\*h\*(w+1));  
 M->width = w+1;  
 M->height = h;  
 w++;  
  
 double \* donor;  
 double \* recipient;  
 for (size\_t ii = 0; ii < h; ++ii) {  
 donor = MATRIX->ptr + (w-1)\*ii;  
 recipient = M->ptr + w\*ii;  
 if (i)  
 memcpy(recipient, donor, sizeof(double)\*i);  
 memcpy(recipient+i, m + ii, sizeof(double));  
 memcpy(recipient+i+1, donor+i, sizeof(double)\*(w-i-1));  
 }  
  
 printMatrix(M);  
 getchar();  
 return M;  
}  
  
void insert(Matrix \*\* MATRIX, size\_t \* nom\_) {  
 size\_t nom = \*nom\_;  
 getchar();  
 char \* answer = NULL;  
 printf("What would you like to insert?\n");  
 printf("1 - column || 2 - row\n->");  
 answer = get\_line();  
 while (is\_digit(answer) == 0) {  
 free(answer);  
 printf("-> ");  
 answer = get\_line();  
 }  
 int n = atoi(answer);  
 free(answer);  
  
 if (n < 1 || n > 2) {  
 printf("Your number is out of range!\n");  
 sleep(2);  
 return;  
 }  
   
 printf("Which Matrix would you like to change?\nEnter index of it: ");  
 answer = get\_line();  
 while (is\_digit(answer) == 0) {  
 free(answer);  
 printf("-> ");  
 answer = get\_line();  
 }  
 size\_t ind = atol(answer);  
 free(answer);  
  
 if (ind < 0 || ind > nom-1) {  
 printf("Your index is out of range!\n");  
 sleep(2);  
 return;  
 }  
  
 printf("Where do you wanna insert your this?\nEnter index of it: ");  
 answer = get\_line();  
 while (is\_digit(answer) == 0) {  
 free(answer);  
 printf("-> ");  
 answer = get\_line();  
 }  
 size\_t ind\_n = atol(answer);  
 free(answer);  
  
 size\_t width = (\*MATRIX)[ind].width, height = (\*MATRIX)[ind].height;  
 printf("This Matrix's size: %ld x %ld.\n", height, width);  
  
 if (ind\_n < 0 || (ind\_n > width && n == 1) || (ind\_n > height && n == 2)) {  
 printf("Your index is out of range!\n");  
 sleep(2);  
 return;  
 }  
  
  
 if (n == 2) {  
 double \* m = (double\*) malloc(sizeof(double) \* width);  
 // ввод новой строки  
 printf("Enter your row here: ");  
 char \* line = get\_line();  
 while (line == NULL) {  
 free(line);  
 printf("-> ");  
 line = get\_line();  
 }  
  
 char \*\* words = NULL;  
 size\_t now = 0;  
 words = (char\*\*) malloc(sizeof(char\*));  
 words[0] = strtok(line, " ");  
 now++;  
 while(words[now-1]) {  
 words = (char\*\*) realloc(words, sizeof(char\*)\*(now+1));  
 words[now] = strtok(NULL, " ");  
 now++;  
 }  
  
 now--;  
 words = (char\*\*) realloc(words, sizeof(char\*)\*(now));  
  
 if (now != width) {  
 printf("Enter ERROR!\n");  
 sleep(3);  
 free(words);  
 free(line);  
 free(m);  
 return;  
 }  
  
 //преобразуем строки в числа  
 for (size\_t k = 0; k < width; ++k)  
 m[k] = atof(words[k]);  
 free(words);  
 free(line);  
   
 Matrix\* M = insert\_row((\*MATRIX)+ind, m, ind\_n);  
 free((\*MATRIX)[ind].ptr);  
 (\*MATRIX)[ind] = \*M;  
 free(M);  
 free(m);  
 }  
 else {  
 double \* m = (double\*) malloc(sizeof(double) \* height);  
 // ввод новой строки  
 printf("Enter your column here:\n");  
  
 for (size\_t i = 0; i < height; ++i) {  
 printf("-> ");  
 answer = get\_line();  
 while (is\_float(answer) == 0) {  
 free(answer);  
 printf("retry -> ");  
 answer = get\_line();  
 }  
 m[i] = atof(answer);  
 free(answer);  
 }  
  
 for (size\_t i = 0; i < height; ++i) {  
 printf("-- %lf --\n", m[i]);  
 }  
  
 Matrix\* M = insert\_column((\*MATRIX)+ind, m, ind\_n);  
 free((\*MATRIX)[ind].ptr);  
 (\*MATRIX)[ind] = \*M;  
 free(M);  
 free(m);  
 //\*nom\_ += height;  
 }  
}  
  
Matrix \*delete\_row(Matrix \* MATRIX, size\_t ind\_n) {  
 size\_t w = MATRIX->width, h = MATRIX->height;  
 Matrix \* M = malloc(sizeof(Matrix));  
 M->ptr = malloc(sizeof(double)\*(h-1)\*w);  
 M->width = w;  
 M->height = h-1;  
 h--;  
  
 double \* donor = MATRIX->ptr;  
 double \* recipient = M->ptr;  
 memmove(recipient, donor, w\*ind\_n\*sizeof(double));  
 donor = MATRIX->ptr + w\*(ind\_n+1);  
 recipient = M->ptr + w\*ind\_n;  
 memmove(recipient, donor, ((1+h)\*w - w\*(1+ind\_n))\*sizeof(double));  
 printMatrix(M);  
 return M;  
}  
  
Matrix \* delete\_column(Matrix \* MATRIX, size\_t ind\_n) {  
 double \* donor = MATRIX->ptr + 1+ind\_n;  
 size\_t w = MATRIX->width, h = MATRIX->height;  
 Matrix \* M = (Matrix\*) malloc(sizeof(Matrix));  
 M->ptr = (double\*) malloc(sizeof(double)\*h\*(w-1));  
 M->width = w-1;  
 w--;  
 M->height = h;  
 for (size\_t i = 0; i < h; ++i) {  
 donor = MATRIX->ptr + (w+1)\*i;  
 double \* recipient = M->ptr + w\*i;  
 if (ind\_n != 0)  
 memcpy(recipient, donor, ind\_n\*sizeof(double));  
 donor = MATRIX->ptr + (w+1)\*i+ind\_n+1;  
 recipient = M->ptr + w\*i + ind\_n;  
 memcpy(recipient, donor, (w-ind\_n)\*sizeof(double));  
 }  
  
 printMatrix(M);  
 return M;  
}  
  
void delete(Matrix \*\* MATRIX, size\_t \* nom\_) {  
 size\_t nom = \*nom\_;  
 getchar();  
 char \* answer = NULL;  
 printf("What would you like to insert?\n");  
 printf("1 - column || 2 - row\n->");  
 answer = get\_line();  
 while (is\_digit(answer) == 0) {  
 free(answer);  
 printf("-> ");  
 answer = get\_line();  
 }  
 int n = atoi(answer);  
 free(answer);  
  
 if (n < 1 || n > 2) {  
 printf("Your number is out of range!\n");  
 sleep(2);  
 return;  
 }  
  
 printf("Which Matrix would you like to change?\n-> ");  
 answer = get\_line();  
 while (is\_digit(answer) == 0) {  
 free(answer);  
 printf("-> ");  
 answer = get\_line();  
 }  
 size\_t ind = atol(answer);  
 free(answer);  
  
 if (ind < 0 || ind > nom-1) {  
 printf("Your index is out of range!\n");  
 sleep(2);  
 return;  
 }  
  
 size\_t width = (\*MATRIX)[ind].width, height = (\*MATRIX)[ind].height;  
 printf("This Matrix's size: %ld x %ld.\n", height, width);  
 printMatrix(MATRIX[ind]);  
  
 if (n == 1) {  
 if (width == 1) {  
  
 }  
 printf("Which column would you like to delete?\n-> ");  
 answer = get\_line();  
 while (is\_digit(answer) == 0) {  
 free(answer);  
 printf("-> ");  
 answer = get\_line();  
 }  
 long int ind\_n = atol(answer);  
 free(answer);  
  
 if (ind\_n < 0 || (ind\_n > width - 1 && n == 1) || (ind\_n > height - 1 && n == 2)) {  
 printf("Your index is out of range!\n");  
 sleep(2);  
 return;  
 }  
  
 Matrix \* M = delete\_column(((\*MATRIX)+ind), ind\_n);  
 free((\*MATRIX)[ind].ptr);  
 (\*MATRIX)[ind] = \*M;  
 free(M);  
 }  
 else if (n == 2) {  
 printf("Which row would you like to delete? (enter index)\n-> ");  
 answer = get\_line();  
 while (is\_digit(answer) == 0) {  
 free(answer);  
 printf("-> ");  
 answer = get\_line();  
 }  
 size\_t ind\_n = atol(answer);  
 free(answer);  
  
 if (ind\_n < 0 || (ind\_n > width - 1 && n == 1) || (ind\_n > height - 1 && n == 2)) {  
 printf("Your index is out of range!\n");  
 sleep(2);  
 return;  
 }  
  
 Matrix \* M = delete\_row((\*MATRIX)+ind, ind\_n);  
 free((\*MATRIX)[ind].ptr);  
 (\*MATRIX)[ind].ptr = M->ptr;  
 (\*MATRIX)[ind].height--;  
 free(M);  
 }  
 answer = get\_line();  
 free(answer);  
}  
  
double getMatrixIJ(Matrix\* m, size\_t i, size\_t j) {  
 return m->ptr[m->width\*i+j];  
}  
  
void setMatrixIJ(Matrix\* m, size\_t i, size\_t j, double val) {  
 m->ptr[m->width\*i+j] = val;  
}  
  
Matrix \*getMinorMatrix(Matrix\* m, int i, int j) {  
 if (m->width != m->height)  
 return NULL;  
 if (i < 0 || i >= m->width)  
 return NULL;  
 if (j < 0 || j >= m->width)  
 return NULL;  
  
 Matrix\* res = malloc(sizeof(Matrix));  
 res->height = m->height - 1;  
 res->width = m->width - 1;  
 res->ptr = malloc(sizeof(double)\*res->height\*res->width);  
  
 for (int n = 0; n < res->width; n++) {  
 double\* from = m->ptr + (n >= i ? n + 1 : n)\*m->width;  
 double\* to = res->ptr + n\*res->width;  
 if (j != 0) {  
 memcpy(to, from, j\*sizeof(double));  
 }  
 if (j != m->height - 1) {  
 memcpy(to + j, from + j + 1, sizeof(double)\*(res->width - j));  
 }  
 }  
 return res;  
}  
  
void removeMatrix(Matrix\* m) {  
 free(m->ptr);  
 free(m);  
}  
  
double determinant(Matrix\* m) {  
 if (m->width == 1)  
 return m->ptr[0];  
  
 double res = 0;  
 int sign = 1;  
 for (int i = 0; i < m->width; i++) {  
 Matrix\* minor = getMinorMatrix(m, 0, i);  
 res += (sign \* determinant(minor) \* getMatrixIJ(m, 0, i));  
 removeMatrix(minor);  
 sign \*= -1;  
 }  
 return res;  
}  
  
void getAllMinors(Matrix\* m) {  
 for (int i = 0; i < m->width; i++) {  
 for (int j = 0; j < m->width; j++) {  
 Matrix\* minor = getMinorMatrix(m, i, j);  
 printMatrix(minor);  
 printf("det: %lf\n\n", determinant(minor));  
 removeMatrix(minor);  
 }  
 }  
}  
  
void all\_minors(Matrix \* MATRIX, size\_t nom) {  
 size\_t ind = 0;  
 //getchar();  
 printf("Which Matrix would you like to change?\n-> ");  
 char \* answer = get\_line();  
 while (is\_digit(answer) == 0) {  
 free(answer);  
 printf("-> ");  
 answer = get\_line();  
 }  
 ind = atol(answer);  
 free(answer);  
 //getchar();  
  
 size\_t width = MATRIX[ind].width, height = MATRIX[ind].height;  
 //getchar();  
 printf("The %ld matrix:\n", ind+1);  
 printf("SIZE: %ldx%ld\n", height, width);  
 for (size\_t j = 0; j < height; ++j) {  
 for (size\_t k = 0; k < width; ++k)  
 printf("%5.lf", MATRIX[ind].ptr[j\*MATRIX[ind].width+k]);  
 printf("\n");  
 }  
  
 getAllMinors(MATRIX + ind);  
 answer = get\_line();  
 free(answer);  
}

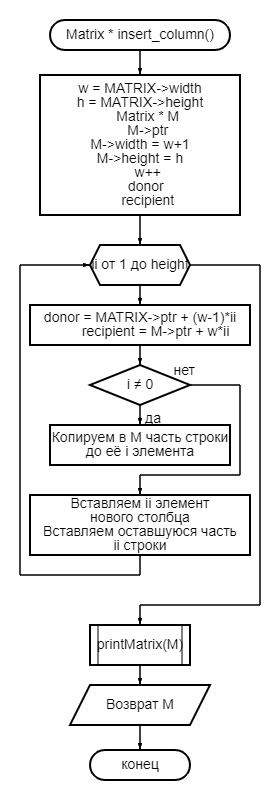
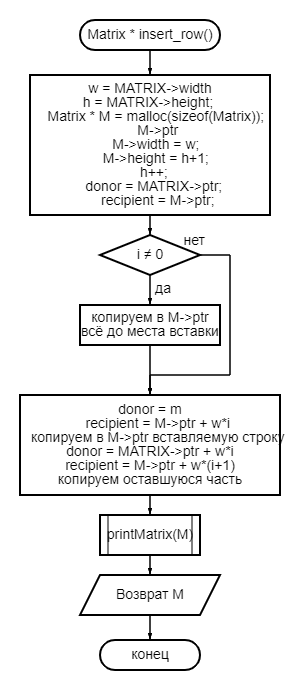
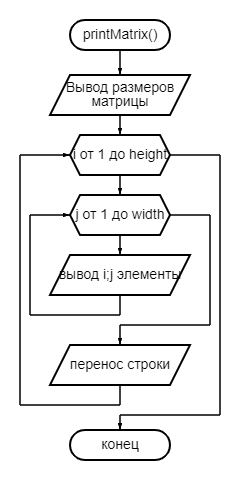
**Тесты.**

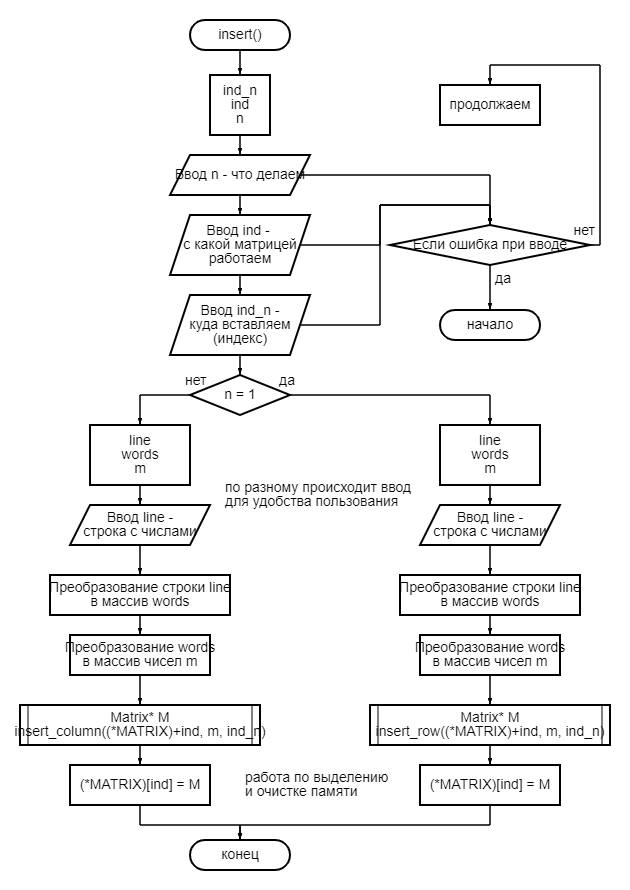
Сгенерировать матрицу 5х5 и найти её миноры.

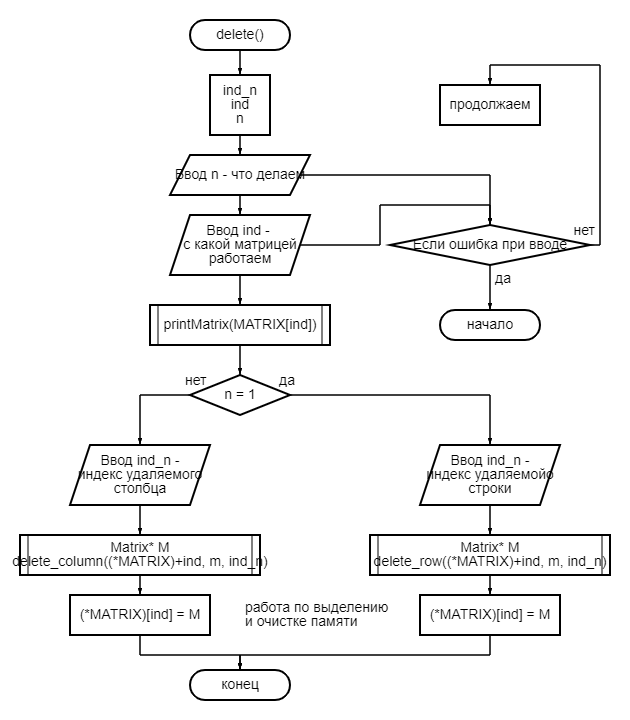
**Скриншоты.**

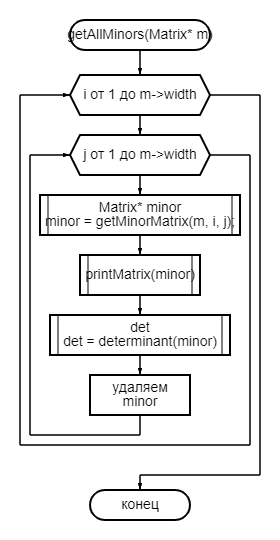
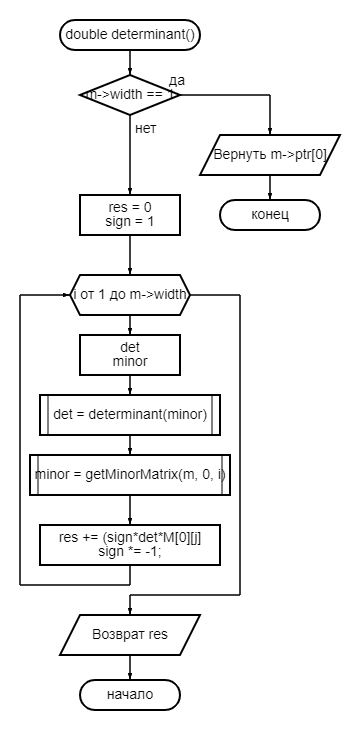
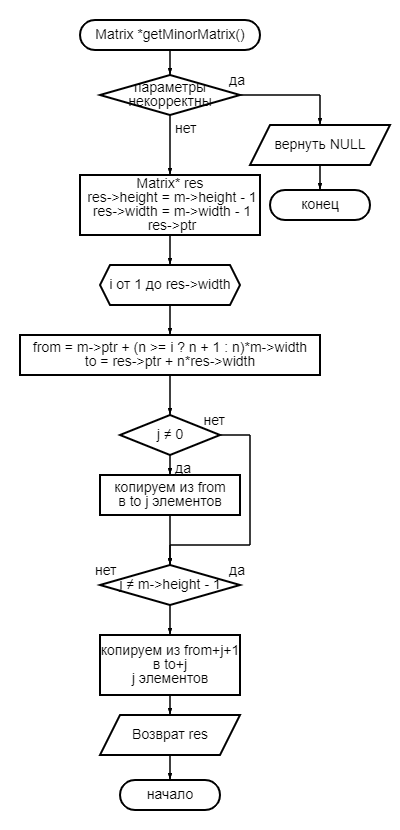


**Блоксхемы функций обработки.**









**Выводы:**

В ходе данной лабораторной работы – БДЗ были закреплены навыки работы со структурами и изучена работа с матрицами в языке си (лишь один из вариантов работы).

Написано несколько довольно трудоёмких функций.

Изучена возможность группировки нескольких заголовочных файлов в одну папку для удобства пользования.

Самостоятельно изучены азы работы с библиотеками .so расширения.