**ЗАДАНИЕ**

Написать программу для обработки матриц. Матрицы хранится в двух разных файлах.

Нужно реализовать следующие операции:

вычисление суммы, разности, произведения двух матриц, а также поиск обратной матрицы от одной из выбранных

**РЕШЕНИЕ**

#include <stdio.h>

#include <stdlib.h>

void number\_input(char\* str, int\* p) { // функция присвоит необходимое(числовое) значение переменной в основной функции, находящейся по адресу(2-ой аргумент)

char string[256];

printf("%s", str);

fgets(string, sizeof(string), stdin);

while (sscanf(string, "%i", p) != 1) {

printf("Incorrect input, try again!! %s", str);

fgets(string, sizeof(string), stdin);

}

}

void fill\_matrix(double\*\* M, int str, int col, char\* string, char\* ch) { // функция заполнения пользователем матрицы

printf("\n%s\n", string);

for (int i = 0; i < str; i++) {

for (int j = 0; j < col; j++) {

printf("%s[%d][%d] = ", ch, i, j);

scanf("%lf", &M[i][j]);

}

}

}

void output\_matrix(double\*\* M, int str, int col, char\* string, char\* ch) { // функция выводит значения матрицы

printf("\n %s\n", string);

for (int i = 0; i < str; i++) {

for (int j = 0; j < col; j++) {

printf("%s[%d][%d] = %.2lf ", ch, i, j, M[i][j]);

}

printf("\n");

}

}

/\*void sum\_matriсes(double\*\* A, double\*\* B, int str, int col) { // функция для сложения вычитания матриц через параметры

double\*\* M = malloc(str \* sizeof(double\*));

for (int i = 0; i < str; i++) {

M[i] = malloc(col \* sizeof(double));

}

int choise;

getchar();

do {

number\_input("\nInput number of operation with matrixes ~ (1: A - B; 2: A + B; 3: B - A) -> ", &choise);

} while (choise != 1 && choise != 2 && choise != 3);

for (int i = 0; i < str; i++) {

for (int j = 0; j < col; j++) {

if (choise == 1)

M[i][j] = A[i][j] - B[i][j];

else if (choise == 2)

M[i][j] = A[i][j] + B[i][j];

else

M[i][j] = B[i][j] - A[i][j];

}

}

if (choise == 1)

output\_matrix(M, str, col, "Values of matrix (A - B):", "M");

else if (choise == 2)

output\_matrix(M, str, col, "Values of matrix (A + B):", "M");

else

output\_matrix(M, str, col, "Values of matrix (B - A):", "M");

}\*/

int sum\_matriсes\_from\_files() { // функция суммирует или вычитает матрицы, считывая их из файлов, а не получая через аргументы

int str1, col1, str2, col2;

FILE\* fp = fopen("add1\_A\_matrix.bin", "rb"); // открываем файл(хранящий первую матрицу) для чтения

fscanf(fp, "%d %d", &str1, &col1); // считываем количество строк/столбцов первой матрицы

double\*\* A = malloc(str1 \* sizeof(double\*));

for (int i = 0; i < str1; i++) {

A[i] = malloc(col1 \* sizeof(double));

for (int j = 0; j < col1; j++)

fscanf(fp, " %lf", &A[i][j]); // считываем элементы матрицы из файла

}

fclose(fp);

fp = fopen("add1\_B\_matrix.bin", "rb"); // теперь считываем вторую матрицу из её файла

fscanf(fp, "%d %d", &str2, &col2);

double\*\* B = malloc(str2 \* sizeof(double\*));

for (int i = 0; i < str2; i++) {

B[i] = malloc(col2 \* sizeof(double));

for (int j = 0; j < col2; j++)

fscanf(fp, " %lf", &B[i][j]);

}

fclose(fp);

if (str1 != str2 || col1 != col2) { // проверяем, что матрицы имеют одинаковый размер

printf("\nIt is impossible to sum up the matrices\n");

getchar();

return 1;

}

double\*\* M = malloc(str1 \* sizeof(double\*)); // выделяем память под матрицу, которая будет хранить сумму наших матриц

for (int i = 0; i < str1; i++) {

M[i] = malloc(col1 \* sizeof(double));

}

int choise; // переменная будет отвечать за номер операции, которую выберет пользователь

getchar();

do {

number\_input("\nInput number of operation with matrixes ~ (1: A + B; 2: A - B; 3: B - A) -> ", &choise);

} while (choise != 1 && choise != 2 && choise != 3);

for (int i = 0; i < str1; i++) {

for (int j = 0; j < col1; j++) {

if (choise == 1) // записываем в матрицу М значения в зависимости от пользовательского выбора

M[i][j] = A[i][j] + B[i][j];

else if (choise == 2)

M[i][j] = A[i][j] - B[i][j];

else

M[i][j] = B[i][j] - A[i][j];

}

}

if (choise == 1) // выводим нашу матрицу суммы

output\_matrix(M, str1, col1, "Values of matrix (A + B):", "M");

else if (choise == 2)

output\_matrix(M, str1, col1, "Values of matrix (A - B):", "M");

else

output\_matrix(M, str1, col1, "Values of matrix (B - A):", "M");

return 0;

}

void multiply\_matriсes(double\*\* A, double\*\* B, int str1, int col1, int str2, int col2) { // функция перемножает матрицы

int choise;

double\*\* M; // матрица будет хранить значение перемножения матриц

do {

number\_input("\nInput number of operation with matrixes ~ (1: A \* B; 2: B \* A) -> ", &choise);

} while (choise != 1 && choise != 2);

if (choise == 1 && col1 == str2) { // в зависимости от выбранной операции проверяем матрицы на возможность перемножения

int str\_col = col1;

M = malloc(str1 \* sizeof(double\*));

for (int i = 0; i < str1; i++) {

M[i] = calloc(col2, sizeof(double));

}

for (int i = 0; i < str1; i++) { // создаём матрицу М

for (int j = 0; j < col2; j++) {

for (int k = 0; k < str\_col; k++) {

M[i][j] += A[i][k] \* B[k][j];

}

}

}

output\_matrix(M, str1, col2, "Values of matrix (A \* B):", "M"); // выводим её

}

else if (choise == 1)

printf("\nIt is impossible to multiply the matrices (A \* B)\n");

if (choise == 2 && col2 == str1) { // аналогично для второго случая

int str\_col = col2;

M = malloc(str2 \* sizeof(double\*));

for (int i = 0; i < str2; i++) {

M[i] = calloc(col1, sizeof(double));

}

for (int i = 0; i < str2; i++) {

for (int j = 0; j < col1; j++) {

for (int k = 0; k < str\_col; k++) {

M[i][j] += B[i][k] \* A[k][j];

}

}

}

output\_matrix(M, str2, col1, "Values of matrix (B \* A):", "M");

}

else if (choise == 2)

printf("\nIt is impossible to multiply the matrices (B \* A)\n");

}

void inverse\_matrix(double\*\* M, int str\_col, char\* string) { // функция получает матрицу, обратную данной, но только если она квадратная

double\*\* M\_1 = malloc(str\_col \* sizeof(double\*));

for (int i = 0; i < str\_col; i++) {

M\_1[i] = calloc(str\_col, sizeof(double));

M\_1[i][i] = 1;

} // создали единичную матрицу

for (int i = 0; i < str\_col; i++) { // с помощью расширенной матрицы и метода Гаусса получаем из единичной матрицы, матрицу обратную данной

double kef = 1 / M[i][i];

for (int k = 0; k < str\_col; k++) {

M[i][k] \*= kef;

M\_1[i][k] \*= kef;

}

for (int j = 0; j < str\_col; j++) {

if (j == i)

continue;

kef = M[j][i];

for (int k = 0; k < str\_col; k++) {

M[j][k] -= kef \* M[i][k];

M\_1[j][k] -= kef \* M\_1[i][k];

}

}

}

output\_matrix(M\_1, str\_col, str\_col, string, "M");

}

int main() {

int str1, col1, str2, col2; // переменные отвечают за количество строк/столбцов двух наших матриц

number\_input("Input amount of strings in first matrix: ", &str1); // пользователь вводит значение количества строк/столбцов первой матрицы

number\_input("Input amount of columns in first matrix: ", &col1);

double\*\* A = malloc(str1 \* sizeof(double\*)); // создаём первую матрицу

for (int i = 0; i < str1; i++)

A[i] = malloc(col1 \* sizeof(double)); // выделили под неё память

fill\_matrix(A, str1, col1, " Input value in first matrix", "A"); // вводим в неё значения

getchar();

number\_input("\nInput amount of strings in second matrix: ", &str2); // теперь заполняем вторую матрицу

number\_input("Input amount of columns in second matrix: ", &col2);

double\*\* B = malloc(str2 \* sizeof(double\*));

for (int i = 0; i < str2; i++)

B[i] = malloc(col2 \* sizeof(double));

fill\_matrix(B, str2, col2, " Input value in second matrix", "B");

FILE\* fp = fopen("add1\_A\_matrix.bin", "wb"); // открываем файл для записи в него первой матрицы

fprintf(fp, "%d %d", str1, col1); // в первую очередь записываем в файл количество строк/столбцов матрицы

for (int i = 0; i < str1; i++)

for (int j = 0; j < col1; j++)

fprintf(fp, " %lf", A[i][j]); // теперь поэлементно записываем матрицу

fclose(fp);

fp = fopen("add1\_B\_matrix.bin", "wb"); // записываем вторую матрицу в файл

fprintf(fp, "%d %d", str2, col2);

for (int i = 0; i < str2; i++)

for (int j = 0; j < col2; j++)

fprintf(fp, " %lf", B[i][j]);

fclose(fp);

output\_matrix(A, str1, col1, "Values of first matrix:", "A"); // выводим матрицы

output\_matrix(B, str2, col2, "Values of second matrix:", "B");

/\*if (str1 == str2 && col1 == col2) // часть кода для сложения матриц через функцию – sum\_matrices(), а не sum\_matriсes\_from\_files()

sum\_matriсes(A, B, str1, col1);

else {

printf("\nIt is impossible to sum up the matrices\n");

getchar();

}\*/

sum\_matriсes\_from\_files(); // складываем или вычитаем матрицы

multiply\_matriсes(A, B, str1, col1, str2, col2); // перемножаем матрицы

// выводим обратные матрицы, если они квадратные, иначе подсчёт обратной матрицы сильно усложняется

if (str1 == col1)

inverse\_matrix(A, str1, "Values of matrix inverse to A:");

else

printf("\nIt is hard find matrix inverse to A\n");

if (str2 == col2)

inverse\_matrix(B, str2, "Values of matrix inverse to B:");

else

printf("\nIt is hard find matrix inverse to B\n");

return 0;

}

**ТЕСТЫ**

**Тест № 1** // пример содержит все возможные операции с матрицами одновременноИзображение выглядит как текст

Автоматически созданное описание

**Тест № 2** // случай, когда ни одна из операций над матрицами не работает

**Изображение выглядит как текст

Автоматически созданное описание**

**Тест № 3** // работает только сложение матриц**Изображение выглядит как текст

Автоматически созданное описание**

**Тест № 4** // возможно только умножениеИзображение выглядит как текст

Автоматически созданное описание

**Тест № 5** // возможно только изъятие обратных матриц