**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: Матрицы и указатели.

|  |  |  |
| --- | --- | --- |
| Студент гр. 3372 |  | Беляев К. В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Научиться работать с матрицами и указателями

Приложение А

рабочий код

#include <iostream>  
#include <limits>  
#include <iomanip>  
#include <stdlib.h>  
  
**using namespace** std;  
**int** line\_down\_times;  
**const int** ORDER = 6, SIZE = ORDER \* ORDER, block\_order = ORDER / 2;  
**int** \*pos1, \*pos2, \*pos3, \*pos4;  
  
  
**void** lineDown(**int** times=1){  
 **for**(line\_down\_times = 0; line\_down\_times < times; line\_down\_times++){  
 cout << "\n";  
 }  
}  
  
  
**void** printMenu(){  
 cout << "1. Заполнение матрицы";  
 lineDown();  
 cout << "2. Перестановка блоков матрицы";  
 lineDown();  
 cout << "3. Сортировка матрицыматрицы";  
 lineDown();  
 cout << "4. Мат. операции с матрицей";  
 lineDown();  
 cout << "5. ИДЗ 10. Определитель матрицы(3х3)";  
 lineDown(2);  
}  
  
  
**void** printIdzMatrix(**int const** n, **int** mtx[n][n]){  
 **int** size = (n \* n) - 1;  
  
 **for** (**int** \*ptr = \*mtx; ptr <= \*mtx + size; ptr++){  
 cout << setw(4) << \*ptr;  
 **if** ((ptr - \*mtx + 1) % n == 0){  
 lineDown();  
 }  
 }  
 lineDown();  
}  
  
  
**void** printMatrix(**int** mtx[][ORDER]){  
 **int** size = (ORDER \* ORDER) - 1;  
  
 **for** (**int** \*ptr = \*mtx; ptr <= \*mtx + size; ptr++){  
 cout << setw(4) << \*ptr;  
 **if** ((ptr - \*mtx + 1) % ORDER == 0){  
 lineDown();  
 }  
 }  
 lineDown();  
}  
  
  
**void** clearStream(){  
 cin.clear();  
 cin.ignore(numeric\_limits<streamsize>::max(), '\n');  
}  
  
  
**int** randint(**int** end){  
 **return** (rand() % end) + 1;  
}  
  
  
**void** moveMatrixBlocks(**int** mtx[][ORDER], **int** method) {  
 **for** (**int** i = 0; i < block\_order; i++) {  
 **for** (**int** j = 0; j < block\_order; j++) {  
 pos1 = \*(mtx + i) + j;  
 pos2 = pos1 + block\_order;  
 pos3 = pos1 + ((ORDER + 1) \* block\_order);  
 pos4 = pos1 + (ORDER \* block\_order);  
 **if** (method == 1){  
 swap(\*pos1, \*pos2);  
 swap(\*pos1, \*pos3);  
 swap(\*pos1, \*pos4);  
  
 } **else if** (method == 2){  
 swap(\*pos1, \*pos3);  
 swap(\*pos2, \*pos4);  
  
 } **else if** (method == 3) {  
 swap(\*pos1, \*pos4);  
 swap(\*pos2, \*pos3);  
  
 } **else if** (method == 4){  
 swap(\*pos1, \*pos2);  
 swap(\*pos3, \*pos4);  
 }  
 }  
 }  
}  
  
  
**void** fillMatrix(**int** mtx[ORDER][ORDER], **int** method){  
 **if** (method == 1)  
 {  
 **int** N = ORDER;  
 **int** i = 0;  
 **int** j = 0;  
 **for** (**int** n = 0; n < ORDER / 2; n++){  
  
 **for** (;j < N; j++){ //right  
 \*(\*(mtx + i) + j) = randint(SIZE);  
 }  
 j--;  
 i++;  
  
 **for** (;i < N; i++){ // down  
 \*(\*(mtx + i) + j) = randint(SIZE);  
 }  
 i--;  
 j--;  
  
 **for** (;j > ORDER - N - 1; j--){ // left  
 \*(\*(mtx + i) + j) = randint(SIZE);  
 }  
 j++;  
 i--;  
 N--;  
  
 **for** (;i > ORDER - N - 1; i--){ // up  
 \*(\*(mtx + i) + j) = randint(SIZE);  
 }  
 j = n + 1;  
 i = n + 1;  
  
 }  
  
 }  
 **else if** (method == 2)  
 {  
 **int** \*ptr = \*mtx;  
  
 **for** (**int** col = 1; col <= ORDER; col++){  
 **if** (col % 2 ){ // down  
 **for** (**int** row = 0; row < ORDER; row++){  
 \*ptr = randint(SIZE);  
 ptr += ORDER;  
 }  
 ptr -= ORDER;  
  
 }**else**{ // up  
 **for** (**int** row = 0; row < ORDER; row++){  
 \*ptr = randint(SIZE);  
 ptr -= ORDER;  
 }  
 ptr += ORDER;  
 }  
 ptr++;  
 }  
 }  
}  
  
  
**void** varyPlus(**int** mtx[][ORDER], **int** coef){  
  
 **for** (**int** \*ptr = \*mtx; ptr < \*mtx + (ORDER\*ORDER); ptr++){  
 \*ptr += coef;  
 }  
}  
  
  
**void** varyMult(**int** mtx[][ORDER], **float** coef){  
 **for** (**int** \*ptr = \*mtx; ptr < \*mtx + (ORDER\*ORDER); ptr++){  
 \*ptr \*= coef;  
 }  
}  
  
  
**int** findPivot(**int** mtx[][ORDER], **int** min, **int** max) {  
 **int** pivot = min - 1;  
 **int** \*ptr = \*mtx;  
 **for** (**int** i = min; i < max; ++i){  
 **if**(\*(ptr + i) < \*(ptr + max)) {  
 pivot++;  
 swap(\*(ptr + i), \*(ptr + max));  
 }  
 }  
 pivot++;  
 swap(\*(ptr + pivot), \*(ptr + max));  
 **return** pivot;  
}  
  
  
**void** quickSort(**int** mtx[][ORDER], **int** min, **int** max) {  
 **if** (min >= max) {  
 **return**;  
 }  
 **int** pivot = findPivot(mtx, min, max);  
 quickSort(mtx, min, pivot - 1);  
 quickSort(mtx, pivot + 1, max);  
}  
  
  
**void** shakerSort(**int** mtx[][ORDER]){  
 **for** (**int** i = 0; i < SIZE / 2; i++){  
 **for** (**int** j = i; j < SIZE - i - 1; j++){  
 **if** (\*(\*mtx + j) > \*(\*mtx + j + 1)){  
 swap(\*(\*mtx + j), \*(\*mtx + j + 1));  
 }  
 }  
 **for** (**int** j = SIZE - i - 2; j > i ; j--){  
 **if** (\*(\*mtx + j) < \*(\*mtx + j - 1)){  
 swap(\*(\*mtx + j), \*(\*mtx + j - 1));  
 }  
 }  
 }  
}  
  
  
**void** fillIdzMatrix(**int** mtx[3][3]){  
 **for** (**int** \*ptr = \*mtx; ptr < \*mtx + (3 \* 3); ptr++){  
 \*(ptr) = rand() % 60 - 29;  
 }  
}  
  
  
**int** findDet(**int** mtx[3][3]){  
 **int** slag = 0;  
 **int** det = 0;  
 **int** \*ptr = \*mtx;  
 slag = (\*(ptr) \* \*(ptr + 4) \* \*(ptr + 8)) + (\*(ptr + 1) \* \*(ptr + 5) \* \*(ptr + 6)) + (\*(ptr + 3) \* \*(ptr + 7) \* \*(ptr + 2));  
 det += slag;  
 slag = (\*(ptr + 2) \* \*(ptr + 4) \* \*(ptr + 6)) + (\*(ptr) \* \*(ptr + 5) \* \*(ptr + 7)) + (\*(ptr + 1) \* \*(ptr + 3) \* \*(ptr + 8));  
 det -= slag;  
 **return** det;  
  
}  
  
  
**void** printIdzMatrix(**int const** n, **int** mtx[3][3]){  
 **int** size = (n \* n) - 1;  
  
 **for** (**int** \*ptr = \*mtx; ptr <= \*mtx + size; ptr++){  
 cout << setw(4) << \*ptr;  
 **if** ((ptr - \*mtx + 1) % n == 0){  
 lineDown();  
 }  
 }  
 lineDown();  
}  
  
  
**int** main(){  
 **int** mtx[ORDER][ORDER], idz\_mtx[3][3];  
 **int** problem;  
 **int** method;  
 **float** coef;  
 **int** det;  
  
 printMenu();  
  
 **for**(**int** repeat = 0; repeat < 100; ++repeat) {  
 cout << "Введите номер задания: ";  
 cin >> problem;  
 clearStream();  
  
 **switch** (problem) {  
 **case** 1:  
 cout << "Введите способ заполнения матрицы(1 или 2): ";  
 cin >> method;  
 clearStream();  
 fillMatrix(mtx, method);  
 printMatrix(mtx);  
 **break**;  
  
 **case** 2:  
 cout << "Введите способ перестановки блоков: ";  
 cin >> method;  
 clearStream();  
 moveMatrixBlocks(mtx, method);  
 printMatrix(mtx);  
 **break**;  
  
 **case** 3:  
 shakerSort(mtx);  
 printMatrix(mtx);  
 **break**;  
  
 **case** 4:  
 cout << "Введите число на которое все элементы матриц +, -, / и \* : ";  
 cin >> coef;  
  
 cout << "+" << coef;  
 lineDown();  
 varyPlus(mtx, coef);  
 printMatrix(mtx);  
 varyPlus(mtx, -coef);  
  
 cout << "-" << coef;  
 lineDown();  
 varyPlus(mtx, -coef);  
 printMatrix(mtx);  
 varyPlus(mtx, coef);  
  
 cout << "\*" << coef;  
 lineDown();  
 varyMult(mtx, coef);  
 printMatrix(mtx);  
 varyMult(mtx, 1.0 / coef);  
  
 cout << "/" << coef;  
 lineDown();  
 varyMult(mtx, 1.0 / coef);  
 printMatrix(mtx);  
 varyMult(mtx, coef);  
 **break**;  
  
 **case** 5:  
 fillIdzMatrix(idz\_mtx);  
 printIdzMatrix(3, idz\_mtx);  
 det = findDet(idz\_mtx);  
 cout << "det = " << det;  
 lineDown();  
 **break**;  
 }  
 }  
  
 **return** 0;  
}