**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №1**

**по дисциплине «Программирование»**

Тема: Двумерные статические массивы. Указатели.

|  |  |  |
| --- | --- | --- |
| Студент гр. 4372 |  | Федюшкин Л.В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

 Используя арифметику указателей, заполнить квадратичную целочисленную матрицу; Переставить ее блоки в соответствии со схемами; Используя арифметику указателей сортировать массив; Уменьшить, увеличить, умножить или делить все элементы матрицы на введенное пользователем число.

**Основные теоретические положения.**

Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения. Указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя.

Присвоение указателю адреса уже рассматривалось в прошлой теме. Для получения адреса объекта используется операция &:

int a = 1917;

int \*pa = &a; // указатель pa хранит адрес переменной a

При этом указатель и переменная должны иметь один и тот же тип, в данном случае это тип int.

Присвоение указателю другого указателя:

#include <iostream>

using std::cout;

using std::endl;

int main()

{

int a = 10;

int b = 2;

int \*pa = &a;

int \*pb = &b;

cout << "Variable a: address=" << pa << "\t value=" << \*pa << endl;

cout << "Variable b: address=" << pb << "\t value=" << \*pb << endl;

pa = pb; // теперь указатель pa хранит адрес переменной b

cout << "Variable b: address=" << pa << "\t value=" << \*pa << endl;

return 0;

}

Когда указателю присваивается другой указатель, то фактически первый указатель начинает также указывать на тот же адрес, на который указывает второй указатель.

**Операция разыменования** указателя представляет выражение в виде \*имя\_указателя. Эта операция позволяет получить объект по адресу, который хранится в указателе.

#include <iostream>

using std::cout;

using std::endl;

int main()

{

int a = 10;

int \*pa = &a;

int \*pb = pa;

\*pa = 25;

cout << "Value on pointer pa: " << \*pa << endl; // 25

cout << "Value on pointer pb: " << \*pb << endl; // 25

cout << "Value of variable a: " << a << endl; // 25

return 0;

}

Через выражение \*pa мы можем получить значение по адресу, который хранится в указателе pa, а через выражение типа \*pa = значение вложить по этому адресу новое значение.

И так как в данном случае указатель pa указывает на переменную a, то при изменении значения по адресу, на который указывает указатель, также изменится и значение переменной a.

Указателю, имеющему такой же базовый тип, как и элементы массива, можно присвоить массив следующим образом:

int Arr[10];

int \*p;

p = Arr;

**Постановка задачи.**

Используя арифметику указателей, заполнить квадратичную целочисленную матрицу; Переставить ее блоки в соответствии со схемами; Используя арифметику указателей сортировать массив; Уменьшить, увеличить, умножить или делить все элементы матрицы на введенное пользователем число.

**Выполнение работы.**

|  |
| --- |
| Меню выбора задания: |
| При выборе 1 пункта открывается меню выбора схемы заполнения матрицы: |
| Заполнение матрицы: |
| Во 2 пункте открывается меню выбора схемы преставления блоков: |
| В 3 пункте выводится сортированный массив: |
| В 4 пункте у пользователя есть выбор, что сделать с матрицей: |
| Пример вывода матрицы при делении на 0; |
| ИДЗ |

**Выводы.**

Используя арифметику указателей, заполнить квадратичную целочисленную матрицу; Переставить ее блоки в соответствии со схемами; Используя арифметику указателей сортировать массив; Уменьшить, увеличить, умножить или делить все элементы матрицы на введенное пользователем число.

Приложение А

рабочий код

#include <iostream>

#include <windows.h>

using namespace std;

//Палитра цветов

namespace Color {

// ANSI escape codes для изменения цвета текста

const string RESET = "\033[0m"; //сброс

const string RED = "\033[1m\033[31m"; //красный

const string GREEN = "\033[1m\033[32m";//зеленый

const string YELLOW = "\033[1m\033[33m"; //желтый

const string BLUE = "\033[1m\033[34m";//синий

const string MAGENTA = "\033[1m\033[35m";// пурпурный

const string CYAN = "\033[1m\033[36m";// голубой

const string BLACK = "\033[1m\033[30m";// черный

const string WHITE = "\033[1m\033[37m"; // белый

void Red(const string& text) {

cout << RED << text << RESET << endl;

}

void Green(const string& text) {

cout << GREEN << text << RESET << endl;

}

void Yellow(const string& text) {

cout << YELLOW << text << RESET << endl;

}

void Blue(const string& text) {

cout << BLUE << text << RESET << endl;

}

void Magenta(const string& text) {

cout << MAGENTA << text << RESET << endl;

}

void Cian(const string& text) {

cout << CYAN << text << RESET << endl;

}

void Black(const string& text) {

cout << BLACK << text << RESET << endl;

}

void White(const string& text) {

cout << WHITE << text << RESET << endl;

}

}

using namespace Color;

const int rows = 6; // Количество строк

const int cols = 6; // Количество столбцов

void bubbleSort(int (\*arr)[cols], int rows) {

int Elements = rows \* cols;

for (int i = 0; i < Elements - 1; ++i)

{

for (int j = 0; j < Elements - i - 1; ++j)

{

int\* current = \*arr + j;

int\* next = \*arr + j + 1;

if (\*current > \*next)

{

swap(\*current, \*next);

}

}

}

}

void fillSpiral(int arr[rows][cols]) {

int\* end = \*(arr + rows \* cols - 1);

for (int\* ptr = \*arr; ptr <= end; ptr++)

{

\*ptr = 1 + rand() % (rows \* cols);

}

}

void printSpiral(int (\*arr)[cols]) {

HANDLE hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD coord;

int top = 0, bottom = rows - 1;

int left = 0, right = cols - 1;

while (top <= bottom && left <= right) {

// Вывод верхней строки

for (int i = left; i <= right; i++) {

coord.X = i \* 3;

coord.Y = top;

SetConsoleCursorPosition(hStdout, coord);

cout << \*(\*(arr + top) + i) << " ";

Sleep(150);

}

top++;

// Вывод правого столбца

for (int i = top; i <= bottom; i++) {

coord.X = right \* 3;

coord.Y = i;

SetConsoleCursorPosition(hStdout, coord);

cout << \*(\*(arr + i) + right) << " ";

Sleep(150);

}

right--;

// Вывод нижней строки

if (top <= bottom) {

for (int i = right; i >= left; i--) {

coord.X = i \* 3;

coord.Y = bottom;

SetConsoleCursorPosition(hStdout, coord);

cout << \*(\*(arr + bottom) + i) << " ";

Sleep(150);

}

bottom--;

}

// Вывод левого столбца

if (left <= right) {

for (int i = bottom; i >= top; i--) {

coord.X = left \* 3;

coord.Y = i;

SetConsoleCursorPosition(hStdout, coord);

cout << \*(\*(arr + i) + left) << " ";

Sleep(150);

}

left++;

}

}

coord.X = 0;

coord.Y = rows + 1;

SetConsoleCursorPosition(hStdout, coord);

}

void printZmeika(int (\*arr)[cols]) {

HANDLE hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD coord;

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

if (col % 2 == 0) {

for (int row = 0; row < rows; row++) {

coord.X = col \* 3;

coord.Y = row;

SetConsoleCursorPosition(hStdout, coord);

cout << \*(\*(arr + row) + col) << " ";

Sleep(150);

}

}

else {

for (int row = rows - 1; row >= 0; row--) {

coord.X = col \* 3;

coord.Y = row;

SetConsoleCursorPosition(hStdout, coord);

cout << \*(\*(arr + row) + col) << " ";

Sleep(150);

}

}

}

coord.X = 0;

coord.Y = rows + 1;

SetConsoleCursorPosition(hStdout, coord);

}

void change\_A(int (\*arr)[cols])

{

int blockSize = rows / 2;

int temp[rows][cols];

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

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

// Блок 1 (верхний левый) -> Блок 2 (верхний правый)

\*(\*(temp + i) + (j + blockSize)) = \*(\*(arr + i) + j);

// Блок 2 (верхний правый) -> Блок 3 (нижний правый)

\*(\*(temp + (i + blockSize)) + (j + blockSize)) = \*(\*(arr + i) + (j + blockSize));

// Блок 3 (нижний правый) -> Блок 4 (нижний левый)

\*(\*(temp + (i + blockSize)) + j) = \*(\*(arr + (i + blockSize)) + (j + blockSize));

// Блок 4 (нижний левый) -> Блок 1 (верхний левый)

\*(\*(temp + i) + j) = \*(\*(arr + (i + blockSize)) + j);

}

}

// Копируем новые значения обратно в оригинальную матрицу

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

{

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

{

\*(\*(arr + i) + j) = \*(\*(temp + i) + j);

}

}

}

void change\_B(int (\*matrix)[cols]) {

int blockSize = rows / 2;

int temp[rows][cols];

// Блок 1 -> Блок 4

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

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

\*(\*(temp + i + blockSize) + (j + blockSize)) = \*(\*(matrix + i) + j); // Блок 1 в Блок 4

}

}

// Блок 2 -> Блок 3

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

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

\*(\*(temp + i + blockSize) + j) = \*(\*(matrix + i) + (j + blockSize)); // Блок 2 в Блок 3

}

}

// Блок 3 -> Блок 2

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

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

\*(\*(temp + i) + (j + blockSize)) = \*(\*(matrix + (i + blockSize)) + j); // Блок 3 в Блок 2

}

}

// Блок 4 -> Блок 1

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

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

\*(\*(temp + i) + j) = \*(\*(matrix + (i + blockSize)) + (j + blockSize)); // Блок 4 в Блок 1

}

}

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

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

\*(\*(matrix + i) + j) = \*(\*(temp + i) + j);

}

}

}

void change\_C(int (\*matrix)[cols]) {

int blockSize = rows / 2;

int temp[rows][cols];

// Блок 1 -> Блок 3

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

{

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

{

\*(\*(temp + i + blockSize) + j) = \*(\*(matrix + i) + j);

}

}

// Блок 3 -> Блок 1

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

{

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

{

\*(\*(temp + i) + j) = \*(\*(matrix + (i + blockSize)) + j);

}

}

// Блок 2 -> Блок 4

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

{

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

{

\*(\*(temp + i) + (j + blockSize)) = \*(\*(matrix + (i + blockSize)) + (j + blockSize));

}

}

// Блок 4 -> Блок 2

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

{

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

{

\*(\*(temp + i + blockSize) + (j + blockSize)) = \*(\*(matrix + i) + (j + blockSize));

}

}

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

{

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

{

\*(\*(matrix + i) + j) = \*(\*(temp + i) + j);

}

}

}

void change\_D(int (\*matrix)[cols]) {

int blockSize = rows / 2;

int temp[rows][cols];

// Блок 1 -> Блок 2

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

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

\*(\*(temp + i) + (j + blockSize)) = \*(\*(matrix + i) + j);

}

}

// Блок 2 -> Блок 1

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

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

\*(\*(temp + i) + j) = \*(\*(matrix + i) + (j + blockSize));

}

}

// Блок 3 -> Блок 4

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

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

\*(\*(temp + (i + blockSize)) + (j + blockSize)) = \*(\*(matrix + (i + blockSize)) + j);

}

}

// Блок 4 -> Блок 3

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

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

\*(\*(temp + (i + blockSize)) + j) = \*(\*(matrix + (i + blockSize)) + (j + blockSize));

}

}

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

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

\*(\*(matrix + i) + j) = \*(\*(temp + i) + j);

}

}

}

void printMatrix(int (\*matrix)[cols])

{

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

{

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

{

cout << \*(\*(matrix + i) + j) << " ";

}

cout << endl;

}

}

void printMatrixA(int (\*matrix)[cols], int a) {

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

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

cout << \*(\*(matrix + i) + j) + a << " ";

}

cout << "\n";

}

}

void printMatrixB(int (\*matrix)[cols], int a) {

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

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

cout << \*(\*(matrix + i) + j) - a << " ";

}

cout << "\n";

}

}

void printMatrixC(int (\*matrix)[cols], int a) {

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

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

cout << \*(\*(matrix + i) + j) \* a << " ";

}

cout << "\n";

}

}

void printMatrixD(int (\*matrix)[cols], int a) {

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

{

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

cout << \*(\*(matrix + i) + j) / a << " ";

}

cout << endl;

}

}

void printMatrixD\_1(float (\*matrix)[cols], int a)

{

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

{

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

{

if (matrix[i][j] > 0)

{

float pop = 1.0;

cout << pop / a << " ";

}

else

{

float pop = -1.0;

cout << pop / a << " ";

}

}

cout << endl;

}

}

//ИДЗ##########################################

void fillSpiral\_2(int arr[3][3])

{

int\* end = \*(arr + rows \* cols - 1);

for (int\* ptr = \*arr; ptr <= end; ptr++)

{

\*ptr = rand()%61 - 30;

}

}

void printMatrix\_2(int (\*matrix)[3])

{

for (int i = 0; i < 3; ++i)

{

for (int j = 0; j < 3; ++j)

{

cout << \*(\*(matrix + i) + j) << " ";

}

cout << endl;

}

}

int main()

{

setlocale(LC\_ALL, "RU");

srand(time(NULL));

Blue("Чтобы выйти из какого - либо пункта вводить 0");

int array[rows][cols];

fillSpiral(array);

while (true)

{

//Меню

cout << "\033[1m\033[31m";

Red("Выберите пункт (1-4) или 0 для выхода\n");

cout << "\033[0m";

cout << "1. Пункт 1. Заполнение матрицы\n";

cout << "2. Пункт 2. Переставление блоков \n";

cout << "3. Пункт 3. Сортировка\n";

cout << "4. Пункт 4. Работа с элементами матрицы\n";

cout << "5. ИДЗ найти определитель матрицы 3\*3\n";

cout << "0. Выход\n";

int choice;

cin >> choice;

if (choice == 0) {

cout << "\nПрограмма завершила свою работу" << std::endl;

break;

}

switch (choice)

{

case 1:

while (true)

{

cout << "\033[1m\033[31m";

Green("\nВы выбрали пункт 1");

cout << "\033[0m";

Blue("Выберите какую схему заполнения матрицы вы хотите увидеть: ");

cout << "1. Спиральная\n";

cout << "2. Змейка\n";

int exs;

cout << "Введите число: ";

cin >> exs;

switch (exs)

{

case 1:

system("cls");

printSpiral(array);

break;

case 2:

system("cls");

printZmeika(array);

break;

default:

break;

}

if (exs == 0)

break;

}

case 2:

while (true)

{

cout << "\033[1m\033[31m";

Green("\nВы выбрали пункт 2");

cout << "\033[0m";

Blue("Выберите какую схему переставления блоков вы хотите увидеть: ");

cout << "1. a\n";

cout << "2. b\n";

cout << "3. c\n";

cout << "4. d\n";

int j;

cout << "Введите число: ";

cin >> j;

switch (j)

{

case 1:

system("cls");

Cian("Схема A");

cout << "Матрица: " << endl;

printMatrix(array);

cout << endl;

cout << "Матрица после изменений: " << endl;

change\_A(array);

printMatrix(array);

break;

case 2:

system("cls");

Cian("Схема B");

cout << "Матрица: " << endl;

printMatrix(array);

cout << endl;

cout << "Матрица после изменений: " << endl;

change\_B(array);

printMatrix(array);

break;

case 3:

system("cls");

Cian("Схема C");

cout << "Матрица: " << endl;

printMatrix(array);

cout << endl;

cout << "Матрица после изменений: " << endl;

change\_C(array);

printMatrix(array);

break;

case 4:

system("cls");

Cian("Схема D");

cout << "Матрица: " << endl;

printMatrix(array);

cout << endl;

cout << "Матрица после изменений: " << endl;

change\_D(array);

printMatrix(array);

break;

default:

break;

}

if (j == 0)

break;

}

case 3:

Green("Несортированный массив: ");

printMatrix(array);

Green("Cортированный массив: ");

bubbleSort(array, cols);

printMatrix(array);

case 4:

while (true)

{

float matrix1[rows][cols];

for (int p = 0; p < rows;p++)

{

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

{

matrix1[p][j] = array[p][j];

}

}

cout << "\033[1m\033[31m";

Green("\nВы выбрали пункт 4");

cout << "\033[0m";

Blue("Выберите какое действие с матрицей вы хотите выполнить: ");

cout << "1. Увеличить на число a каждый элемент\n";

cout << "2. Уменьшить на число a каждый элемент\n";

cout << "3. Умножить на число a каждый элемент\n";

cout << "4. Поделить на число a каждый элемент\n";

int p;

cout << "Введите число: ";

cin >> p;

int a;

cout << "Введите число для работы с матрицей: ";

cin >> a;

switch (p)

{

case 1:

system("cls");

Cian("Увеличить на a");

Yellow("Матрица: ");

printMatrix(array);

cout << endl;

Yellow("Матрица после изменений: ");

printMatrixA(array, a);

break;

case 2:

system("cls");

Cian("Уменьшить на a");

Yellow("Матрица: ");

printMatrix(array);

cout << endl;

Yellow("Матрица после изменений: ");

printMatrixB(array, a);

break;

case 3:

system("cls");

Cian("Умножить на a");

Yellow("Матрица: ");

printMatrix(array);

cout << endl;

Yellow("Матрица после изменений: ");

printMatrixC(array, a);

break;

case 4:

system("cls");

Cian("Поделить на a");

Yellow("Матрица: ");

printMatrix(array);

cout << endl;

Yellow("Матрица после изменений: ");

printMatrixD\_1(matrix1, a);

break;

default:

break;

}

if (p == 0)

break;

}

case 5:

int matrica[3][3];

fillSpiral\_2(matrica);

system("cls");

Cian("Исходная матрица: ");

printMatrix\_2(matrica);

Green("Нахождение определителя методом треугольника: ");

Yellow("Схема вычисления определителя: ");

cout << "(a11\*a22\*a33)+(a12\*a23\*a31)+(a13\*a21\*a32) - ((a11\*a23\*a32) + (a12\*a21\*a33) + (a13\*a22\*a31))"<<endl;

Yellow("Промежуточные вычисления : ");

cout << "(" << matrica[0][0] \* matrica[1][1] \* matrica[2][2]

<< ") + (" << matrica[0][1] \* matrica[1][2] \* matrica[2][0]

<< ") + (" << matrica[0][2] \* matrica[1][0] \* matrica[2][1]

<< ") - ((" << matrica[0][0] \* matrica[1][2] \* matrica[2][1]

<< ") + (" << matrica[0][1] \* matrica[1][0] \* matrica[2][2]

<< ") + (" << matrica[0][2] \* matrica[1][1] \* matrica[2][0] << "))"<< endl;

cout << "Определитель матрицы = " << (matrica[0][0] \* matrica[1][1] \* matrica[2][2]

) + (matrica[0][1] \* matrica[1][2] \* matrica[2][0]

) + (matrica[0][2] \* matrica[1][0] \* matrica[2][1]

) - ((matrica[0][0] \* matrica[1][2] \* matrica[2][1]

) + ( matrica[0][1] \* matrica[1][0] \* matrica[2][2]

) + ( matrica[0][2] \* matrica[1][1] \* matrica[2][0]))<<endl;

}

}

return 0;

}