**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

Тема: **ДВУМЕРНЫЕ СТАТИЧЕСКИЕ МАССИВЫ. УКАЗАТЕЛИ**

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. | Минин М. А. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2022

**Цель работы.**

Изучение двумерных статических массивов: объявление массивов, ввод-вывод массивов; получение практических навыков работы с двумерными статическими массивами, указателями.

**Основные теоретические положения.**

Массив представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение. Каждый элемент в этих массивах определяется значением индекса элемента.

Все массивы можно разделить на две группы: одномерные и многомерные. Многомерные массивы определяются аналогично одномерным массивам. Количество элементов по каждому измерению указывается отдельно в квадратных скобках. Для доступа к определенному элементу многомерного массива необходимо указать в квадратных скобках конкретные значения всех индексов этого элемента.

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных. Так же, как и в одномерном массиве, элементы многомерных массивов располагаются друг за другом в непрерывном участке памяти.

Значения индексов элементов массивов всегда начинается с 0. Поэтому максимальное значение индекса элемента в массиве всегда на единицу меньше количества элементов в массиве.

Ввод и вывод массивов реализуются с помощью циклов.

Указатели – это обычные переменные, но они служат для хранения адресов памяти.

Указатели определяются в программе следующим образом:

<тип данных>\* <имя переменной>

Формально указатели представляют собой обычные целые значения типа int и занимают в памяти 4 байта не зависимо от базового типа указателя. Значения указателей при их выводе на экран представляются как целыезначения в шестнадцатеричном формате.

Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения.

**Постановка задачи.**

1. Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка N (6,8,10) случайными числами от 1 до N\*N согласно схемам, приведенным на рисунках (спираль, змейка). Пользователь должен видеть процесс заполнения квадратичной матрицы.
2. Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами.
3. Используя арифметику указателей, сортирует элементы любой сортировкой.
4. Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

**Выполнение работы.**

Код программы представлен в приложении А.

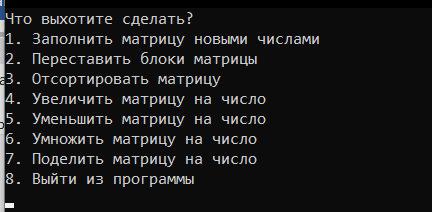
1. При запуске программы пользователю выводится меню доступных команд и ожидается ввод команды с клавиатуры.

Рис. 1 – Меню программы

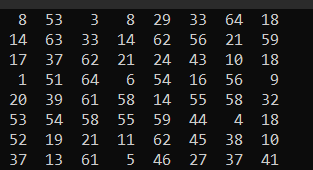
1. Вывод матрицы спиралью или змейкой

Рис. 2 – Вывод матрицы

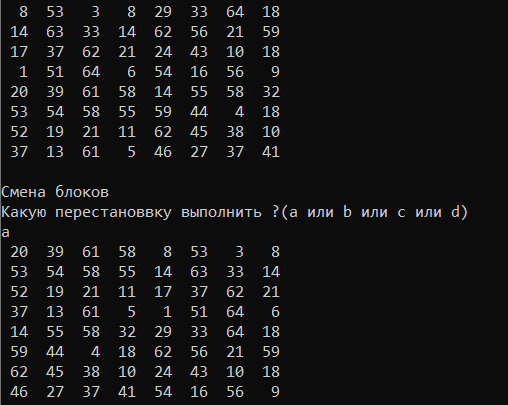
1. Перестановка блоков

Рис.3 – Перестановка блоков

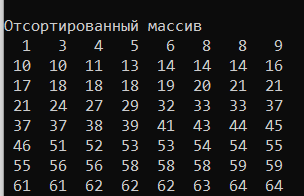
1. Сортировка матрицы

Рис.4 – Сортировка матрицы

**Вывод.**

В ходе работы было изучено: объявление двумерных статических массивов, ввод-вывод массивов, работа с указателями, связь массивом и указателей.

Приложение А

рабочий код

#include <iostream>

#include <iomanip>

#include <windows.h>

using namespace std;

void PrintMatrix(int\*\* Matrix, int Size) {

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

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

cout << setw(3) << setprecision(3) << Matrix[i][j];

cout.flush();

if (j != Size)

cout << ' ';

}

cout << "\n";

}

Sleep(50);

}

void EmptyMatrix(int\*\* Matrix, int Size) {

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

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

Matrix[i][j] = 0;

}

void SpiralMatrix(int\*\* Matrix, int Size) {

EmptyMatrix(Matrix, Size);

int LineNumber = 0;

for (int LineSize = 0; LineSize < Size / 2; LineSize++) {

for (int\* i = \*(Matrix + LineNumber) + LineNumber, \*End = i + Size - 2 \* LineNumber; i < End; i++) {

\*i = rand() % (Size \* Size) + 1;

system("cls");

PrintMatrix(Matrix, Size);

}

for (int\*\* i = Matrix + LineNumber + 1, \*\*End = Matrix + Size - 2 - LineNumber; i <=End; i++) {

\*(\*i + Size - LineNumber - 1) = rand() % (Size \* Size) + 1;

system("cls");

PrintMatrix(Matrix, Size);

}

for (int\* i = \*(Matrix + Size - LineNumber - 1) + Size - LineNumber - 1, \*End = i - Size + 1 + 2 \* LineNumber; i > End; i--) {

\*i = rand() % (Size \* Size) + 1;

system("cls");

PrintMatrix(Matrix, Size);

}

for (int\*\* i = Matrix + Size - 1 - LineNumber, \*\*End = Matrix + LineNumber + 1; i >= End; i--) {

\*(\*i+LineNumber) = rand() % (Size \* Size) + 1;

system("cls");

PrintMatrix(Matrix, Size);

}

LineNumber++;

}

}

void SnakeMatrix(int\*\* Matrix, int Size) {

EmptyMatrix(Matrix, Size);

for (int LineNumber = 0; LineNumber < Size / 2; LineNumber++) {

for (int\*\* i = Matrix; i <= (Matrix + Size - 1); i++) {

\*(\*i + LineNumber \* 2) = rand() % (Size \* Size) + 1;

system("cls");

PrintMatrix(Matrix, Size);

}

for (int\*\* i = Matrix + Size - 1; i >= Matrix; i--) {

\*(\*i + LineNumber \* 2 + 1) = rand() % (Size \* Size) + 1;

system("cls");

PrintMatrix(Matrix, Size);

}

}

}

void RotateMatrix(int\*\* Matrix, int Size) {

int\*\* New\_Matrix = new int\* [Size];

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

New\_Matrix[i] = new int[Size];

char permutation;

cout << "Какую перестановвку выполнить ?(a или b или c или d)" << '\n';

cin >> permutation;

while (permutation != 'a' and permutation != 'b' and permutation != 'c' and permutation != 'd') {

cout << "Неверно введён вид перестановки, попробуйте снова: ";

cin >> permutation;

}

if (permutation == 'a') {

for (int i = 0; i < Size / 2; i++)

for (int j = 0; j < Size / 2; j++) {

New\_Matrix[i][j] = Matrix[i + Size / 2][j];

New\_Matrix[i][j + Size / 2] = Matrix[i][j];

New\_Matrix[i + Size / 2][j + Size / 2] = Matrix[i][j + Size / 2];

New\_Matrix[i + Size / 2][j] = Matrix[i + Size / 2][j + Size / 2];

}

Matrix = New\_Matrix;

PrintMatrix(Matrix, Size);

}

else if (permutation == 'b') {

for (int i = 0; i < Size / 2; i++)

for (int j = 0; j < Size / 2; j++) {

New\_Matrix[i][j] = Matrix[i + Size / 2][j + Size / 2];

New\_Matrix[i][j + Size / 2] = Matrix[i + Size / 2][j];

New\_Matrix[i + Size / 2][j + Size / 2] = Matrix[i][j];

New\_Matrix[i + Size / 2][j] = Matrix[i][j + Size / 2];

}

Matrix = New\_Matrix;

PrintMatrix(Matrix, Size);

}

else if (permutation == 'c') {

for (int i = 0; i < Size / 2; i++)

for (int j = 0; j < Size / 2; j++) {

New\_Matrix[i][j] = Matrix[i + Size / 2][j];

New\_Matrix[i][j + Size / 2] = Matrix[i + Size / 2][j + Size / 2];

New\_Matrix[i + Size / 2][j + Size / 2] = Matrix[i][j + Size / 2];

New\_Matrix[i + Size / 2][j] = Matrix[i][j];

}

Matrix = New\_Matrix;

PrintMatrix(Matrix, Size);

}

else {

for (int i = 0; i < Size / 2; i++)

for (int j = 0; j < Size / 2; j++) {

New\_Matrix[i][j] = Matrix[i][j + Size / 2];

New\_Matrix[i][j + Size / 2] = Matrix[i][j];

New\_Matrix[i + Size / 2][j + Size / 2] = Matrix[i + Size / 2][j];

New\_Matrix[i + Size / 2][j] = Matrix[i + Size / 2][j + Size / 2];

}

Matrix = New\_Matrix;

PrintMatrix(Matrix, Size);

}

}

void SortMatrix(int\*\* Matrix, int Size) {

int ArrSize = Size \* Size;

int\* Array = new int[ArrSize];

int Array\_i = 0;

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

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

Array[Array\_i] = Matrix[i][j];

Array\_i++;

}

}

bool Change = TRUE;

while (Change) {

Change = FALSE;

for (int\* i = Array, \*End\_i = i + ArrSize - 2; i <= End\_i; i++) {

if (\*i > \*(i+1)) {

Change = TRUE;

swap(\*i, \*(i+1));

}

}

}

Array\_i = 0;

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

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

Matrix[i][j] = Array[Array\_i];

Array\_i++;

}

}

PrintMatrix(Matrix, Size);

}

void MagnificationMatrix(int\*\* Matrix, int Size) {

int Number;

cout << '\n' << "Введите число" << '\n';

cin >> Number;

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

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

Matrix[i][j] += Number;

PrintMatrix(Matrix, Size);

}

void ReducationMatrix(int\*\* Matrix, int Size) {

int Number;

cout << '\n' << "Введите число" << '\n';

cin >> Number;

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

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

Matrix[i][j] -= Number;

PrintMatrix(Matrix, Size);

}

void MultiplicationMatrix(int\*\* Matrix, int Size) {

int Number;

cout << '\n' << "Введите число" << '\n';

cin >> Number;

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

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

Matrix[i][j] \*= Number;

PrintMatrix(Matrix, Size);

}

void DivisionMatrix(int\*\* Matrix, int Size) {

int Number;

cout << '\n' << "Введите число" << '\n';

cin >> Number;

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

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

Matrix[i][j] /= Number;

PrintMatrix(Matrix, Size);

}

void PrintMenu() {

system("cls");

cout << "Что выхотите сделать?" << '\n';

cout << "1. Заполнить матрицу новыми числами" << '\n';

cout << "2. Переставить блоки матрицы" << '\n';

cout << "3. Отсортировать матрицу" << '\n';

cout << "4. Увеличить матрицу на число" << '\n';

cout << "5. Уменьшить матрицу на число" << '\n';

cout << "6. Умножить матрицу на число" << '\n';

cout << "7. Поделить матрицу на число" << '\n';

cout << "8. Выйти из программы" << '\n';

}

void Menu(int\*\* Matrix, int Size) {

int Variant;

system("pause");

do {

PrintMenu();

cin >> Variant;

switch (Variant) {

case 1:

char perm;

cout << "В каком виде вывести матрицу?(a или b)" << '\n';

cin >> perm;

while (perm != 'a' and perm != 'b') {

cout << "Неверно введён вид вывода, попробуйте снова: ";

cin >> perm;

}

if (perm == 'a')

SpiralMatrix(Matrix, Size);

else

SnakeMatrix(Matrix, Size);

break;

case 2:

PrintMatrix(Matrix, Size);

cout << '\n' << "Смена блоков" << '\n';

RotateMatrix(Matrix, Size);

break;

case 3:

cout << '\n' << "Отсортированный массив" << '\n';

SortMatrix(Matrix, Size);

break;

case 4:

PrintMatrix(Matrix, Size);

MagnificationMatrix(Matrix, Size);

break;

case 5:

PrintMatrix(Matrix, Size);

ReducationMatrix(Matrix, Size);

break;

case 6:

PrintMatrix(Matrix, Size);

MultiplicationMatrix(Matrix, Size);

break;

case 7:

PrintMatrix(Matrix, Size);

DivisionMatrix(Matrix, Size);

break;

}

if (Variant != 8)

system("pause");

} while (Variant != 8);

}

int main() {

setlocale(0, "");

srand(time(0));

int Size;

cout << "Введите размер матрицы: 6 или 8 или 10:" << "\n";

cin >> Size;

while (Size != 6 and Size != 8 and Size != 10) {

cout << "Неверно введён размер матрицы, попробуйте снова: ";

cin >> Size;

}

int\*\* Matrix = new int\* [Size];

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

Matrix[i] = new int[Size];

EmptyMatrix(Matrix, Size);

Menu(Matrix, Size);

}