**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.

|  |  |  |
| --- | --- | --- |
| Студентка гр. 4372 |  | Хальметова Ю.Р. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

 Используя арифметику указателей, необходимо заполнить квадратичную целочисленную матрицу, после чего переставить блоки этой матрицы в соответствии с заданными схемами. Далее, применяя арифметику указателей, следует отсортировать одномерный массив. В завершение, необходимо провести операции уменьшения, увеличения, умножения или деления всех элементов матрицы на число, введенное пользователем.

**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.**

| Ввод пользователем и обработка данных | | Работа алгоритма и вывод на экран |
| --- | --- | --- |
| Меню | | |
| При запуске программы перед пользователем появляется, выбор задания. | | Меню: |
| Вывод квадратичной целочисленной матрицы порядка N (6,8,10) случайными числами от 1 до  N\*N | | |
| При вводе пользователем корректного значения пункта меню пользователь выбирает способ вывода матрицы | | Заполняет квадратичную матрицу и выводит ее, исходя из выбора пользователя |
| Вывод на экран созданного массива после перестановки блоков . | | |
| При вводе пользователем корректного значения пункта меню пользователь выбирает способ перестановки матрицы | | Переставляет блоки квадратичной матрицы и выводит ее, исходя из выбора пользователя |
| Сортировка массива | | |
| При запуске программы перед пользователем появляется, выбор задания. | Сортирует матрицу и выводит ее на экран | |
| Над каждым элементом матрицы происходит действие исходя из выбора пользователя | | |
| Пользователь выбирает действие, которое будет производиться над каждым элементом матрицы | Изменяет каждый элемент матрицы и выводит ее на экран | |

**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 <unistd.h>

#include <cstdlib>

#include <iomanip>

using namespace std;

const int rows = 6;

const int cols = 6;

void intToFloatMatrix(const int\* intMatrix, float\* floatMatrix, size\_t rows, size\_t cols) {

for (size\_t i = 0; i < rows; ++i) {

for (size\_t j = 0; j < cols; ++j) {

\*(floatMatrix + i \* cols + j) = static\_cast<float>(\*(intMatrix + i \* cols + j));

}

}

}

void plusNumber(int\* arr, int rows, int cols, int n) {

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

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

\*(arr + i \* cols + j) += n;

}

}

}

void minusNumber(int\* arr, int rows, int cols, int n) {

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

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

\*(arr + i \* cols + j) -= n;

}

}

}

void multiplyNumber(int\* arr, int rows, int cols, int n) {

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

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

\*(arr + i \* cols + j) \*= n;

}

}

}

void shareNumber(float\* f\_arr, int rows, int cols, int n) {

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

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

\*(f\_arr + i \* cols + j) /= n;

}

}

cout << "Измененная матрица: " << endl;

cout << fixed << setprecision(1);

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

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

cout << left << setw(10) << \*(f\_arr + i \* cols + j);

usleep(50000);

}

cout << endl;

}

}

void fillSpiral(int\* arr, int rows, int cols) {

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

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

\*(arr + i \* cols + j) = 1 + rand() % (rows \* cols);

}

}

}

void straightArray(int(\*arr)[cols], int\* flatArray) {

int index = 0;

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

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

\*(flatArray + index++) = \*(\*(arr + i) + j);

}

}

}

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void quickSort(int\* arr, int start, int end) {

if (start >= end) return;

int f = start;

int l = end;

int mid = \*(arr + (f + l) / 2);

while (f <= l) {

while (\*(arr + f) < mid) f++;

while (\*(arr + l) > mid) l--;

if (f <= l) {

swap(arr + f, arr + l);

f++;

l--;

}

}

quickSort(arr, start, l);

quickSort(arr, f, end);

}

void insertionSort(int\* arr, int n) {

int i, j, key;

for (i = 1; i < n; i++) {

key = \*(arr + i);

j = i - 1;

while (j >= 0 && \*(arr + j) > key) {

\*(arr + j + 1) = \*(arr + j);

j--;

}

\*(arr + j + 1) = key;

}

}

void combSort(int\* arr, int n) {

float k = 1.247;

int s = n - 1;

while (s >= 1) {

for (int i = 0; i + s < n; i++) {

if (\*(arr + i) > \*(arr + i + s)) {

swap(\*(arr + i), \*(arr + i + s));

}

}

s /= k;

}

}

void cocktailSort(int\* arr, int n) {

bool swapped = true;

int start = 0;

int end = n - 1;

while (swapped) {

swapped = false;

for (int i = start; i < end; i++) {

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

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

swapped = true;

}

}

if (!swapped) {

break;

}

end--;

for (int i = end; i >= start; i--) {

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

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

swapped = true;

}

}

start++;

}

}

void bubbleSort(int\* arr, int n) {

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

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

if (\*(arr + j) > \*(arr + j + 1)) {

swap(\*(arr + j), \*(arr + j + 1));

}

}

}

}

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

int flatArray[rows \* cols];

straightArray(arr, flatArray);

quickSort(flatArray,0, rows \* cols - 1);

int index = 0;

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

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

\*(\*(arr + i) + j) = \*(flatArray + index++);

}

}

}

void setCursorPosition(int x, int y) {

cout << "\033[" << y << ";" << x << "H";

}

void printMatrix(int arr[][cols], int rows, int cols) {

int\* ptr = (int\*)arr;

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

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

cout << \*(ptr + i \* cols + j) << " ";

usleep(50000);

}

cout << endl;

}

}

void printSpiral(int arr[][cols], int rows, int cols) {

system("clear");

int top = 0, bottom = rows - 1;

int left = 0, right = cols - 1;

int\* ptr = (int\*)arr;

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

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

setCursorPosition(i \* 4, top + 1);

cout << \*(ptr + top \* cols + i);

cout.flush();

usleep(300000);

}

top++;

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

setCursorPosition(right \* 4, i + 1);

cout << \*(ptr + i \* cols + right);

cout.flush();

usleep(300000);

}

right--;

if (top <= bottom) {

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

setCursorPosition(i \* 4, bottom + 1);

cout << \*(ptr + bottom \* cols + i);

cout.flush();

usleep(300000);

}

bottom--;

}

if (left <= right) {

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

setCursorPosition(left \* 4, i + 1);

cout << \*(ptr + i \* cols + left);

cout.flush();

usleep(300000);

}

left++;

}

}

setCursorPosition(0, rows + 2);

}

void printSnake(int arr[][cols], int rows, int cols) {

system("clear");

int\* ptr = (int\*)arr;

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

if (j % 2 == 0) {

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

setCursorPosition(j \* 4, i + 1);

cout << \*(ptr + i \* cols + j);

cout.flush();

usleep(150000);

}

} else {

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

setCursorPosition(j \* 4, i + 1);

cout << \*(ptr + i \* cols + j);

cout.flush();

usleep(300000);

}

}

}

setCursorPosition(0, rows + 2);

}

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) {

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

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

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

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

}

}

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

{

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

{

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

}

}

}

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

int blockSize = rows / 2;

int new\_arr[rows][cols];

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

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

\*(\*(new\_arr + (i + blockSize)) + (j + blockSize)) = \*(\*(arr + i) + j);

\*(\*(new\_arr + i) + j) = \*(\*(arr + (i + blockSize)) + (j + blockSize));

\*(\*(new\_arr + (i + blockSize)) + j) = \*(\*(arr + i) + (j + blockSize));

\*(\*(new\_arr + i) + (j + blockSize)) = \*(\*(arr + (i + blockSize)) + j);

}

}

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

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

\*(\*(arr + i) + j) = \*(\*(new\_arr + i) + j);

}

}

}

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

int blockSize = rows / 2;

int new\_arr[rows][cols];

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

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

\*(\*(new\_arr + (i + blockSize)) + j) = \*(\*(arr + i) + j);

\*(\*(new\_arr + i) + j) = \*(\*(arr + (i + blockSize)) + j);

\*(\*(new\_arr + (i + blockSize)) + (j + blockSize)) = \*(\*(arr + i) + (j + blockSize));

\*(\*(new\_arr + i) + (j + blockSize)) = \*(\*(arr + (i + blockSize)) + (j + blockSize));

}

}

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

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

\*(\*(arr + i) + j) = \*(\*(new\_arr + i) + j);

}

}

}

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

int blockSize = rows / 2;

int new\_arr[rows][cols];

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

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

\*(\*(new\_arr + i) + (j + blockSize)) = \*(\*(arr + i) + j);

\*(\*(new\_arr + i) + j) = \*(\*(arr + i) + (j + blockSize));

\*(\*(new\_arr + (i + blockSize)) + (j + blockSize)) = \*(\*(arr + (i + blockSize)) + j);

\*(\*(new\_arr + (i + blockSize)) + j) = \*(\*(arr + (i + blockSize)) + (j + blockSize));

}

}

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

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

\*(\*(arr + i) + j) = \*(\*(new\_arr + i) + j);

}

}

}

int main(){

int arr[rows][cols];

float f\_arr[rows][cols];

system ("clear");

bool flag = true;

while (flag) {

cout << endl;

cout << "Введите номер задания (1-4): ";

int task;

cin >> task;

cout << endl;

switch(task){

case 1:{

cout << "Выберите вывод (1 - спираль, 2 - змейка): ";

int conclusion;

cin >> conclusion;

fillSpiral(&arr[0][0], rows, cols);

if(conclusion == 1){

printSpiral(arr, rows, cols);

} else {

printSnake(arr, rows, cols);

}

break;

}

case 2:{

cout<<"Выберите перестановку(1/2/3/4): ";

int permutation;

cin >> permutation;

if (permutation == 1){

cout << endl << endl;

cout << "Поворот блоков:\n";

usleep(2000000);

change\_A(arr);

printMatrix(arr, rows, cols);

}

if (permutation == 2){

cout << endl << endl;

cout << "Поворот блоков:\n";

usleep(2000000);

change\_B(arr);

printMatrix(arr, rows, cols);

}

if (permutation == 3){

cout << endl << endl;

cout << "Поворот блоков:\n";

usleep(2000000);

change\_C(arr);

printMatrix(arr, rows, cols);

}

if (permutation == 4){

cout << endl;

cout << "Поворот блоков:\n";

usleep(2000000);

change\_D(arr);

printMatrix(arr, rows, cols);

}

break;

}

case 3:{

cout << "Отсортированный массив: "<< endl;

fillSpiralSorted(arr);

printMatrix(arr, rows, cols);

break;

}

case 4:{

cout << "Выберите действие (1 - плюс, 2 - минус, 3 - умножение, 4 - деление): ";

int operation;

cin >> operation;

cout << endl;

if (operation == 1){

cout << "Ввдеите число, на которое увеличится каждое число матрицы: ";

int n;

cin >> n;

plusNumber(&arr[0][0], rows, cols, n);

cout << "Измененная матрица: " << endl;

printMatrix(arr, rows, cols);

}

if (operation == 2){

cout << "Ввдеите число, на которое уменьшится каждое число матрицы: ";

int n;

cin >> n;

minusNumber(&arr[0][0], rows, cols, n);

cout << "Измененная матрица: " << endl;

printMatrix(arr, rows, cols);

}

if (operation == 3){

cout << "Ввдеите число, на которое умножится каждое число матрицы: ";

int n;

cin >> n;

multiplyNumber(&arr[0][0], rows, cols, n);

cout << "Измененная матрица: " << endl;

printMatrix(arr, rows, cols);

}

if (operation == 4){

cout << "Ввдеите число, на которое поделится каждое число матрицы: ";

int n;

cin >> n;

intToFloatMatrix(&arr[0][0], &f\_arr[0][0], rows, cols);

shareNumber(&f\_arr[0][0], rows, cols, n);

}

}

}

}

}