**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №4**

**по дисциплине «Программирование»**

Тема: Двумерные статические массивы. Указатели.

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

Санкт-Петербург

2024

**Цель работы.**

Изучение двумерных статистических массивов, а также ознакомление с такими понятиями, как указатели и ссылки. Приобретение навыков работы с указателями.

**Основные теоретические положения.**

**Массив** представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Все массивы можно разделить на две группы: одномерные и многомерные. Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных. Объявление в программах одномерных массивов выполняется в соответствии со следующим правилом:

**<Базовый тип элементов> <Идентификатор массива>[<Количество элементов>]**

Так, например:

int ArrInt [10], A1 [20];

Обращение к определенному элементу массива осуществляется с помощью указания значения индекса этого элемента:

A1 [4] = -1200;

cout << A1 [4];

При обращении к конкретному элементу массива этот элемент можно рассматривать как обычную переменную, тип которой соответствует базовому типу элементов массива, и осуществлять со значением этого элемента любые операции, которые характерны для базового типа.

При объявлении массива его можно инициализировать определенными значениями:

short S [5] = {1, 4, 9, 16, 25};

Или:

short S [] = {1, 4, 9, 16, 25};

**Указатели и ссылки** ориентированы на прямую работу с памятью компьютера. С помощью этих средств реализуется работа с динамической памятью и динамическими объектами, возвращение из функций измененных данных и многое другое.

Для получения адреса какого-либо программного объекта используется оператор &. **Указатели** – это обычные переменные, но они служат для хранения адресов памяти. Указатели определяются в программе следующим образом:

**<тип данных>\*<имя переменной>**

Здесь **<тип данных>** определяет базовый тип указателя. **<Имя переменной>**  является идентификатором переменной-указателя. Например:

int \*p1;

double \*p2;

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

**Присваивание:** указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя. Например:

int a = 1917;

int \*pa = &a;

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

**Операции сравнения:** к указателям могут применяться операции сравнения >, >=, <, <=, ==, !=. Операции сравнения применяются только к указателям одного типа и к значениям NULL и nullptr.

К указателям можно применят некоторые арифметические операции. К таким операциям относятся: +, -, ++, --. Результаты их выполнения по отношению к указателям существенно отличаются от результатов соответствующих арифметических операций, выполняющихся с обычными числовыми данными.

**Постановка задачи.**

Необходимо разработать алгоритм и написать программу, которая:

1. Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка N (6,8,10) случайными числами от 1 до  N\*N согласно схемам. 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 из списка: shaker sort, comb sort, insert sort, quick sort.
4. Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число

**Выполнение работы.**

Код программы представлен в приложении А.

Описание кода и использованных алгоритмов:

При запуске программы перед пользователем появляется окно, в котором на экран выводится

1. Интерактивная строка, где пользователь может выбрать размер матрицы (6, 8 или 10)
2. Интерактивная строка, где можно выбрать паттерн заполнения (спираль или «змейка»).
3. Заполняемая в соответствии с выбранным паттерном в реальном времени матрица заданного размера.
4. Интерактивное окно, где можно выбрать, каким образом смещать блоки матрицы.
5. Новая матрица с переставленными определенным образом блоками
6. Матрица, отсортированная с помощью алгоритма quick sort.
7. Интерактивное окно, где можно выбрать действия с элементами матрицы
8. Интерактивное окно, где можно задать число, на которое элементы матрицы будут увеличены, уменьшены, умножены или разделены.
9. Матрица, над которой были произведены преобразования.



Рис. 1. Интерактивная строка для ввода размера матрицы

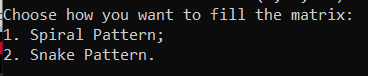


Рис.2 Интерактивная строка, где можно выбрать паттерн заполнения

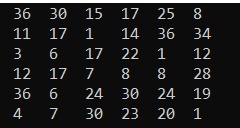


Рис. 3. Заполненная матрица

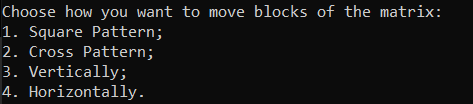


Рис. 4. Интерактивное окно, где можно выбрать, как переместить блоки матрицы.

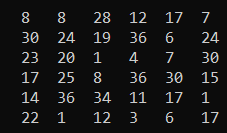


Рис. 5. Матрица с переставленными блоками

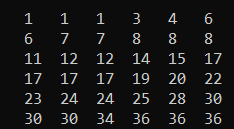


Рис. 6. Матрица, отсортированная с помощью quick sort.

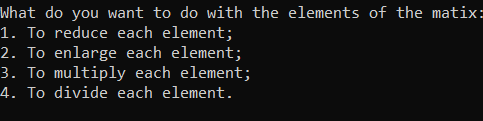


Рис. 7. Интерактивное окно выбора действий над элементами матрицы.



Рис. 8. Интерактивное окно, где можно задать число

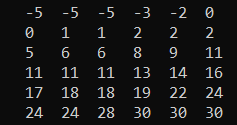


Рис. 9. Измененная матрица

Тестовые данные:

|  |  |
| --- | --- |
| Matrix size | Number |
| 6, 8, 10 | 6, 30, -8 |

**Выводы.**

В ходе работы были изучены двумерные статистические массивы, а также было проведено ознакомление с такими понятиями, как указатели и ссылки. Были приобретены навыки работы с указателями.

Приложение А

рабочий код

#include <iostream>

#include <thread>

#include <chrono>

#include <Windows.h>

using namespace std;

void printMatrix(int\* matrix, int size, int helper) {

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

COORD destCoord;

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

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

destCoord.X = j \* 4 + 50;

destCoord.Y = i + helper;

SetConsoleCursorPosition(hStdout, destCoord);

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

}

cout << endl;

}

}

void quicksort(int\* matrix, int\* end, int\* begin) {

int pivotElement;

int\* start = begin;

int\* finish = end;

pivotElement = \*(begin + (end - begin) / 2);

while (start < finish) {

while (\*start < pivotElement) start++;

while (\*finish > pivotElement) finish--;

if (start <= finish) {

std::swap(\*start, \*finish);

start++;

finish--;

}

}

if (begin < finish) quicksort(matrix, finish, begin);

if (start < end) quicksort(matrix, end, start);

}

void fillSnake(int\* matrix, int n) {

int top = 0, bottom = n - 1, left = 0, right = n - 1;

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

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

int\* num = matrix + i \* n + left;

\*num = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(200);

}

left++;

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

int\* num = matrix + i \* n + left;

\*num = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(200);

}

left++;

}

}

void fillSpiral(int\* matrix, int n) {

int top = 0, bottom = n - 1, left = 0, right = n - 1;

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

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

\*(matrix + top \* n + i) = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(200);

}

top++;

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

\*(matrix + i \* n + right) = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(200);

}

right--;

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

\*(matrix + bottom \* n + i) = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(200);

}

bottom--;

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

\*(matrix + i \* n + left) = rand() % (n \* n) + 1;

printMatrix(matrix, n, 0);

Sleep(200);

}

left++;

}

printMatrix(matrix, n, 0);

}

void swap\_quarters\_a(int\* matrix, int N) {

int\* temp = new int[N \* N];

int middle = N / 2;

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

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

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

}

}

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

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

if (i < middle && j < middle) {

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

}

else if (i < middle && j >= middle) {

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

}

else if (i >= middle && j >= middle) {

\*(matrix + i \* N + (j - middle)) = \*(temp + i \* N + j);

}

else {

\*(matrix + (i - middle) \* N + j) = \*(temp + i \* N + j);

}

}

}

printMatrix(matrix, N, N + 2);

}

void swap\_quarters\_b(int\* matrix, int N) {

int\* temp = new int[N \* N];

int middle = N / 2;

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

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

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

}

}

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

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

if (i < middle && j < middle) {

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

}

else if (i < middle && j >= middle) {

\*(matrix + (i + middle) \* N + j - middle) = \*(temp + i \* N + j);

}

else if (i >= middle && j >= middle) {

\*(matrix + (i - middle) \* N + j - middle) = \*(temp + i \* N + j);

}

else {

\*(matrix + (i - middle) \* N + j + middle) = \*(temp + i \* N + j);

}

}

}

printMatrix(matrix, N, N + 2);

}

void swap\_quarters\_c(int\* matrix, int N) {

int\* temp = new int[N \* N];

int middle = N / 2;

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

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

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

}

}

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

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

if (i < middle) {

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

}

else {

\*(matrix + (i - middle) \* N + j) = \*(temp + i \* N + j);

}

}

}

printMatrix(matrix, N, N + 2);

}

void swap\_quarters\_d(int\* matrix, int N) {

int\* temp = new int[N \* N];

int middle = N / 2;

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

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

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

}

}

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

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

if (j < middle) {

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

}

else {

\*(matrix + i \* N + j - middle) = \*(temp + i \* N + j);

}

}

}

printMatrix(matrix, N, N + 2);

}

void enlarge\_each\_element(int\* matrix, int N) {

int enlarge\_num;

cout << "Enter any number: ";

cin >> enlarge\_num;

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

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

\*(matrix + i \* N + j) += enlarge\_num;

}

}

printMatrix(matrix, N, N + 22);

}

void reduce\_each\_element(int\* matrix, int N) {

int reduce\_num;

cout << "Enter any number: ";

cin >> reduce\_num;

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

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

\*(matrix + i \* N + j) -= reduce\_num;

}

}

printMatrix(matrix, N, N + 22);

}

void divide\_each\_element(int\* matrix, int N) {

int divide\_num;

cout << "Enter any number: ";

cin >> divide\_num;

if (divide\_num == 0) {

cout << "You can not divide by zero" << endl;

return;

}

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

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

\*(matrix + i \* N + j) /= divide\_num;

}

}

printMatrix(matrix, N, N + 22);

}

void multiply\_each\_element(int\* matrix, int N) {

int multiply\_num;

cout << "Enter any number: ";

cin >> multiply\_num;

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

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

\*(matrix + i \* N + j) \*= multiply\_num;

}

}

printMatrix(matrix, N, N + 22);

}

void start(int\* matrix, int size) {

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

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

\*(matrix + i \* size + j) = 1;

}

}

}

int main() {

srand(time(0));

int size, answer, answerFill, answerBlockMove, answerNumber;

cout << "Choose the size of a matrix (6, 8, 10): ";

cin >> size;

int\* matrix = new int[size \* size];

while (true) {

cout << "Choose how you want to fill the matrix: " << endl << "1. Spiral Pattern;" << endl << "2. Snake Pattern." << endl;

cin >> answerFill;

start(matrix, size);

switch (answerFill) {

case 1:

fillSpiral(matrix, size);

break;

case 2:

fillSnake(matrix, size);

break;

}

cout << "Choose how you want to move blocks of the matrix: " << endl << "1. Square Pattern;" << endl << "2. Cross Pattern;" << endl << "3. Vertically; " << endl << "4. Horizontally." << endl;

cin >> answerBlockMove;

switch (answerBlockMove) {

case 1:

swap\_quarters\_a(matrix, size);

break;

case 2:

swap\_quarters\_b(matrix, size);

break;

case 3:

swap\_quarters\_c(matrix, size);

break;

case 4:

swap\_quarters\_d(matrix, size);

break;

}

quicksort(matrix, matrix + size \* size - 1, matrix);

printMatrix(matrix, size, size + 10);

cout << "What do you want to do with the elements of the matix: " << endl << "1. To reduce each element;" << endl << "2. To enlarge each element;" << endl << "3. To multiply each element;" << endl << "4. To divide each element." << endl;

cin >> answerNumber;

switch (answerNumber) {

case 1:

reduce\_each\_element(matrix, size);

break;

case 2:

enlarge\_each\_element(matrix, size);

break;

case 3:

multiply\_each\_element(matrix, size);

break;

case 4:

divide\_each\_element(matrix, size);

break;

}

}

return 0;

}