**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 гр. |  | Полуянов В. Н. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение двумерных статических массив и основных алгоритмов работы с ними. Изучение ссылок и указателей, арифметики указателей. Написание программы, заполняющей матрицу размера N (N = 6, 8, 10) и заполняющий его паттернами «змейка» и «спираль». Программа должна так же менять местами блоки (подмассивы) матрицы в соответствии с четырьмя паттернами обмена из задания. Сортировать матрицу с использованием арифметики указателей и увеличивать, уменьшать, умножать и делить все элементы матрицы на число, введенное пользователем.

**Основные теоретические положения.**

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 другое.

Указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя. Когда указателю присваивается другой указатель, то фактически первый указатель начинает также указывать на тот же адрес, на который указывает второй указатель.

Операция разыменования указателя позволяет получить объект по адресу, который хранится в указателе.

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

**Постановка задачи.**

Разработать алгоритм и написать программу, которая:

1. Создает целочисленную матрицу размерности N\*N и заполняет её нулями.
2. Заполняет матрицу случайными числами от 1 до N^2 змейкой
3. Заполняет матрицу случайными числами от 1 до N^2 по спирали
4. Меняет местами блоки матрицы четырьмя методами, метод выбирает пользователь
5. Сортирует матрицу, используя арифметику указателей
6. Увеличивает, уменьшает, умножает или делит все элементы матрицы на число, которое вводит пользователь. Операция также выбирается пользователем.

**Выполнение работы.**

Код программы представлен в приложении А.

1. При запуске программы пользователю выводится подсказка команды, которая выводит меню доступных команд и ожидается ввод команды с клавиатуры (рис. 1).



Рисунок 1. Запуск программы

1. Следующий шаг зависит от введенной команды, если пользователь ввёл:
   1. “0”, то выполнение программы завершается.
   2. “1”, то массив заполняется с помощью паттерна «Змейка» (рис. 2)

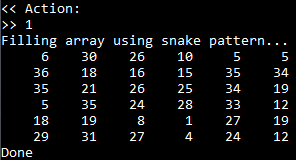


Рисунок 2. Задача 1

* 1. “2”, то массив заполняется с помощью паттерна «Спираль» (рис. 3).

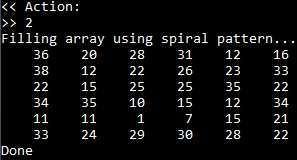


Рисунок 3. Задача 2

* 1. “3”, то ожидается пользовательский выбор паттерна обмена блоков матрицы. После выбора производится обмен и выводятся значения матрицы до и после обмена (рис. 4).

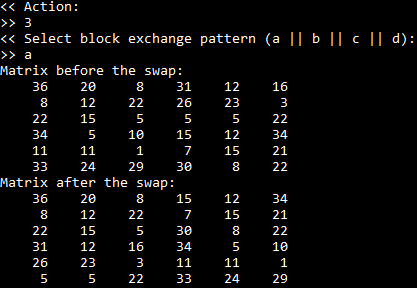


Рисунок 4. Задача 3

* 1. “4”, то выполняется сортировка матрицы сортировкой вставками с использованием арифметики указателей (рис. 5).

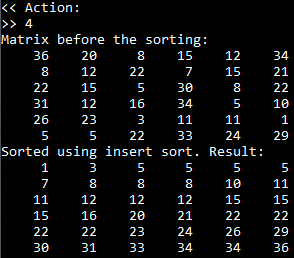


Рисунок 5. Задача 4

* 1. “5”, то ожидается пользовательский выбор операции (сложение, вычитание, умножение или деление) и все элементы увеличиваются, уменьшаются, умножаются или делятся (в зависимости от выбранной операции) на число, которое так же вводит пользователь (рис. 6).

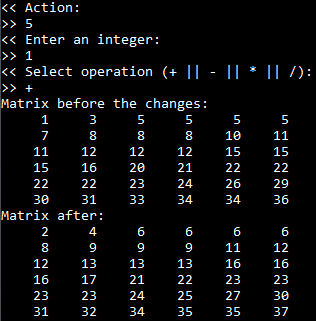


Рисунок 6. Задача 5

* 1. “h”, то выводится меню доступных команд (рис. 7).

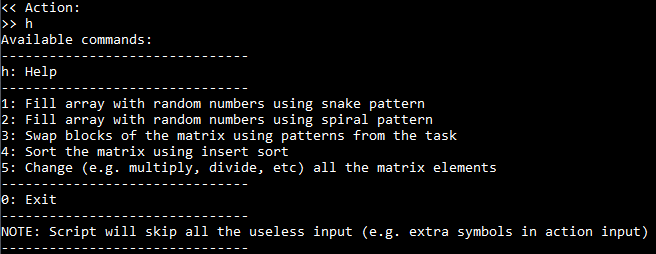


Рисунок 7. Меню

* 1. Любой символ или набор символов, отличный от существующий команд, то будет выведено сообщение об ошибке (рис. 8).



Рисунок 8. Неизвестная команда

**Выводы.**

В ходе работы были изучены двумерные статические массивы и методы работы с ними. Изучены ссылки и указатели, арифметика указателей.

Приложение А

рабочий код

#include <iostream>  
#include <iomanip>  
#include <cstdlib>  
#include <Windows.h>  
  
int getRandInt(int from, int to) {  
 return from + std::rand() % to;  
}  
  
void printMatrix(int \*arr, int N, int width = 6) {  
 for (int i = 0; i < N\*N; i+=N) {  
 for (int j = i; j < i+N; j++) {  
 std::cout.width(width);  
 std::cout << arr[j];  
 }  
 putchar('\n');  
 }  
}  
  
int getIntLength(int x) {  
 if (x == 0) return 1;  
 int amount = 0;  
 while (x) {  
 x /= 10;  
 amount++;  
 }  
 return amount;  
}  
  
int getCursorPositionY() {  
 CONSOLE\_SCREEN\_BUFFER\_INFO screenBufferInfo = {};  
 GetConsoleScreenBufferInfo(GetStdHandle(**STD\_OUTPUT\_HANDLE**), &screenBufferInfo);  
 return screenBufferInfo.dwCursorPosition.Y;  
}  
  
void updateMatrix(int x, int y, int value, int width = 6) {  
 COORD destCoord;  
 destCoord.X = (x+1)\*width - getIntLength(value);  
 destCoord.Y = y;  
  
 SetConsoleCursorPosition(GetStdHandle(**STD\_OUTPUT\_HANDLE**), destCoord);  
  
 std::cout << value << '\r';  
 std::cout.flush();  
  
 Sleep(100);  
}  
  
void zeroFill(int \*arr, int N) {  
 for (int i = 0; i < N\*N; ++i) \*(arr++) = 0;  
}  
  
void randFill(int \*arr, int N, int from, int to) {  
 for (int i = 0; i < N\*N; ++i) \*(arr++) = getRandInt(from, to);  
}  
  
void snakeFill(int \*arr, int N) {  
 *// Init* int startCursorPosY = getCursorPositionY();  
 printMatrix(arr, N);  
  
 *// Main loop* int \*pArr = arr;  
 int change = N;  
 for (int i = 0; i < N; ++i) {  
 int \*pRowEnd = (change > 0) ? pArr + N\*N - N : pArr - N\*N + N;  
 while (pArr - change != pRowEnd) {  
 \*pArr = getRandInt(1, N\*N);  
 updateMatrix((pArr - arr) % N, startCursorPosY + (pArr - arr) / N, \*pArr);  
 pArr += change;  
 }  
 pArr -= change - 1;  
 change = -change;  
 }  
  
 *// Return cursor to the end of the matrix* for (int i = 0; i < N; ++i) putchar('\n');  
}  
  
void spiralFill(int \*arr, int N) {  
 *// Init* int startCursorPosY = getCursorPositionY();  
 printMatrix(arr, N);  
  
 *// Main loop* int \*pArr = arr;  
 int size = N;  
 for (int square = 0; square < N/2; ++square) {  
 int change = 1;  
 for (int direction = 0; direction < 4; ++direction) {  
 for (int i = 0; i < size-1; ++i) {  
 \*pArr = getRandInt(1, N\*N);  
 updateMatrix((pArr - arr) % N, startCursorPosY + (pArr - arr) / N, \*pArr);  
 pArr += change;  
 }  
 change = abs(change) == 1 ? N : 1;  
 change = direction < 1 ? +change : -change;  
 }  
 size -= 2;  
 pArr += N + 1;  
 }  
  
 *// Return cursor to the end of the matrix* for (int i = 0; i < N/2; ++i) putchar('\n');  
}  
  
int \*getQuadPointer(int \*arr, int N, int quadNum) {  
 switch (quadNum) {  
 case 1: return (arr);  
 case 2: return (arr + N/2);  
 case 3: return (arr + N\*N/2);  
 case 4: return (arr + N\*N/2 + N/2);  
 }  
}  
  
void swapQuadrants(int \*arr, int N, int q1, int q2) {  
 *// Get first elements pointers of the quadrants* int \*pQuad1 = getQuadPointer(arr, N, q1);  
 int \*pQuad2 = getQuadPointer(arr, N, q2);  
  
 *// Swap* for (int i = 0; i < N/2; ++i) {  
 for (int j = 0; j < N/2; ++j)  
 std::swap(\*(pQuad1++), \*(pQuad2++));  
 pQuad1 += N - N/2;  
 pQuad2 += N - N/2;  
 }  
}  
  
void insertionSort(int \*arr, int N) {  
 for (int \*p1 = arr+1; p1 < arr + N\*N; ++p1)  
 for (int \*p2 = p1; p2 > arr && \*(p2-1) > \*p2; --p2)  
 std::swap(\*(p2-1), \*p2);  
}  
  
void changeMatrixValues(int \*arr, int N, int operation, int diff) {  
 for (int \*pArr = arr; pArr < arr + N\*N; ++pArr) {  
 switch (operation) {  
 case '+': {  
 \*pArr += diff;  
 break;  
 }  
 case '-': {  
 \*pArr -= diff;  
 break;  
 }  
 case '\*': {  
 \*pArr \*= diff;  
 break;  
 }  
 case '/': {  
 \*pArr /= diff;  
 break;  
 }  
 }  
 }  
}  
  
void subtractMatrix(int \*arr1, int \*arr2, int \*arrRes, int N) {  
 for (int \*p1 = arr1, \*p2 = arr2, \*pR = arrRes; p1 < arr1 + N\*N; ++p1, ++p2, ++pR)  
 \*pR = \*p1 - \*p2;  
}  
  
int main() {  
 *// Init matrix* const int N = 6;  
 int matrix[N][N];  
 int \*ptrMatrix = &matrix[0][0];  
 zeroFill(ptrMatrix, N);  
  
 *// Main loop* std::cout << "Enter 'h' to get list of commands\n";  
 while (true) {  
  
 *// Get command from user* char userAction;  
 std::cout << "<< Action: \n>> ";  
 std::cin.sync();  
 std::cin >> userAction;  
  
 *// Exit* if (userAction == '0') break;  
  
 switch (userAction) {  
  
 *// Fill array using snake pattern* case '1': {  
 std::cout << "Filling array using snake pattern...\n";  
 snakeFill(ptrMatrix, N);  
 std::cout << "Done\n";  
 break;  
 }  
  
 *// Fill array using spiral pattern* case '2': {  
 std::cout << "Filling array using spiral pattern...\n";  
 spiralFill(ptrMatrix, N);  
 std::cout << "Done\n";  
 break;  
 }  
  
 *// Swap blocks* case '3': {  
 *// User input* char userInput;  
 std::cout << "<< Select block exchange pattern (a || b || c || d):\n>> ";  
 std::cin >> userInput;  
 std::cin.sync();  
 if (userInput < 97 || userInput > 100) {  
 std::cout << "Input incorrect. May be another time...\n";  
 break;  
 }  
  
  
 *// Swap* std::cout << "Matrix before the swap:\n";  
 printMatrix(ptrMatrix, N);  
 switch (userInput) {  
 case 'a': {  
 swapQuadrants(ptrMatrix, N, 1, 2);  
 swapQuadrants(ptrMatrix, N, 2, 4);  
 swapQuadrants(ptrMatrix, N, 4, 3);  
 swapQuadrants(ptrMatrix, N, 3, 1);  
 break;  
 }  
 case 'b': {  
 swapQuadrants(ptrMatrix, N, 1, 4);  
 swapQuadrants(ptrMatrix, N, 2, 3);  
 break;  
 }  
 case 'c': {  
 swapQuadrants(ptrMatrix, N, 1, 3);  
 swapQuadrants(ptrMatrix, N, 2, 4);  
 break;  
 }  
 case 'd': {  
 swapQuadrants(ptrMatrix, N, 1, 2);  
 swapQuadrants(ptrMatrix, N, 3, 4);  
 break;  
 }  
 }  
 std::cout << "Matrix after the swap:\n";  
 printMatrix(ptrMatrix, N);  
  
 break;  
 }  
  
 *// Sort the matrix* case '4': {  
 std::cout << "Matrix before the sorting:\n";  
 printMatrix(ptrMatrix, N);  
 insertionSort(ptrMatrix, N);  
 std::cout << "Sorted using insert sort. Result:\n";  
 printMatrix(ptrMatrix, N);  
 break;  
 }  
  
 *// Change all the matrix elements with number entered by user* case '5': {  
 *// User input* int userNumb;  
 std::cout << "<< Enter an integer:\n>> ";  
 std::cin >> userNumb;  
 std::cin.sync();  
 if (std::cin.fail()) {  
 std::cout << "TypeError: invalid literal for int with base 10.\n";  
 std::cin.clear();  
 break;  
 }  
 char userOp;  
 std::cout << "<< Select operation (+ || - || \* || /):\n>> ";  
 std::cin >> userOp;  
 std::cin.sync();  
 if (!(userOp == '+' || userOp == '-' || userOp == '\*' || userOp == '/')) {  
 std::cout << "Input incorrect.\n";  
 break;  
 }  
  
 *// Change* std::cout << "Matrix before the changes:\n";  
 printMatrix(ptrMatrix, N);  
 changeMatrixValues(ptrMatrix, N, userOp, userNumb);  
 std::cout << "Matrix after:\n";  
 printMatrix(ptrMatrix, N);  
 break;  
 }  
  
 *// Individual task №5. Matrix2 - matrix1* case 'i': {  
 *// User input* int size;  
 std::cout << "<< Enter a size of the matrix:\n>> ";  
 std::cin >> size;  
 std::cin.sync();  
 if (std::cin.fail()) {  
 std::cout << "TypeError: invalid literal for int with base 10.\n";  
 std::cin.clear();  
 break;  
 }  
  
 *// Init matrix* int matrix1[size][size], matrix2[size][size], result[size][size];  
 int \*pArr1 = &matrix1[0][0], \*pArr2 = &matrix2[0][0], \*pRes = &result[0][0];  
 randFill(pArr1, size, 1, size\*size);  
 randFill(pArr2, size, 1, size\*size);  
  
 *// Subtract* std::cout << "Subtracting from matrix\n";  
 printMatrix(pArr1, size);  
 std::cout << "Matrix\n";  
 printMatrix(pArr2, size);  
 subtractMatrix(pArr1, pArr2, pRes, size);  
 std::cout << "Result:\n";  
 printMatrix(pRes, size);  
  
 break;  
 }  
  
 *// Help* case 'h': {  
 std::cout << "Available commands:\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "h: Help\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "1: Fill array with random numbers using snake pattern\n";  
 std::cout << "2: Fill array with random numbers using spiral pattern\n";  
 std::cout << "3: Swap blocks of the matrix using patterns from the task\n";  
 std::cout << "4: Sort the matrix using insert sort\n";  
 std::cout << "5: Change (e.g. multiply, divide, etc) all the matrix elements\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "0: Exit\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << "NOTE: Script will skip all the useless input (e.g. extra symbols in action input)\n";  
 std::cout << std::setw(32) << std::setfill('-') << '\n';  
 std::cout << std::setfill(' ');  
 break;  
 }  
  
 *// Unknown command error* default: std::cout << "RuntimeError: unknown command\n";  
 }  
 }  
  
 return 0;  
}