**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. 4373 |  | Репков А.А |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Изучение работы с указателями.

**Основные теоретические положения.**

Указатели и ссылки являются одними из самых важных и достаточно сложных для понимания и использования средств языка программирования. Они 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. К использованию указателей и ссылок мы будем неоднократно возвращаться в последующих разделах.

Для получения адреса какого-либо программного объекта используется оператор **&**. Например, если выполнить фрагмент следующей программы (в предположении, что переменные A и B располагаются в памяти, как это показано на предыдущем рисунке):

int A = 2351;

double B = 3.14;

cout <<  “Значение переменной А: ” << A << endl;

cout <<  “Адрес переменной А: ” << &A << endl;

cout <<  “Значение переменной В: ” << В << endl;

cout <<  “Адрес переменной В: ” << &В << endl;

получим следующий результат:

Значение переменной А: 2351

Адрес переменной А: 102

Значение переменной В: 3.14

Адрес переменной В: 106

Правда, значения адресов переменных будут выведены в шестнадцатеричном  формате.

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

Указатели определяются в программе следующим образом:

**<тип данных> \*<имя переменной>**

Здесь <**тип данных**> определяет так называемый **базовый тип указателя**.

**<Имя переменной>**является идентификатором переменной-указателя.

Признаком того, что это переменная указатель, является символ \*, располагающийся между базовым типом указателя и именем переменной-указателя.

Например:

int \*p1;

double \*p2;

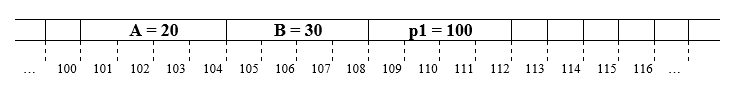
К указателям можно применять некоторые арифметические операции. К таким операциям относятся:  **+**,**-**, **++**, **--**. Результаты выполнения этих операций по отношению к указателям существенно отличаются от результатов соответствующих арифметических операций, выполняющихся с обычными числовыми данными.

Рассмотрим следующий пример:

int A = 20, B = 30;

int \*p1 = &A;

Пусть переменные **A**и **B** расположены в памяти, например, так, как это показано на следующем рисунке:



Указатель **p1** содержит адрес переменной **A**, который равен 100 и \***p1** будет равно значению переменной **A**, то есть 20. Выполним следующую операцию:

p1 = p1 + 1;

или, что то же самое:

p1++;

Значение указателя изменится и станет равным 104, а не 101, как, наверное, ожидалось. То есть теперь указатель ссылается уже на переменную **B**и значение \***p1**будет равно 30.

Таким образом, добавление или вычитание 1 из указателя приводит к изменению его значения на размер базового типа указателя. В общем случае, например, при выполнении следующей операции:

p1 = p1 + N;  //  N – некоторое целое значение

значение указателя увеличится на **sizeof(<базовый тип указателя>) \* N** и в нашем случае это приращение будет равно **sizeof(int) \* N = 4 \* N**. Так, если N = 4, а p1= 100, то значение указателя **p1** увеличится на 16 и станет равно 116, и указатель будет  ссылаться на данные, расположенные по адресу 116.

**Постановка задачи.**

Необходимо 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, приведенным на рисунках. Пользователь должен видеть процесс заполнения квадратичной матрицы (\*Для манипуляции с элементами используйте только арифметику указателей):



2)    Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами (\*Для манипуляции с элементами используйте только арифметику указателей):



3)    Используя арифметику указателей, сортирует элементы любой сортировкой из списка ниже (если во 2 ПР вы реализовывали одну из представленных сортировок, рекомендуется переиспользовать написанный код и модернизировать его для работы с указатями).   
Варианты сортировок:

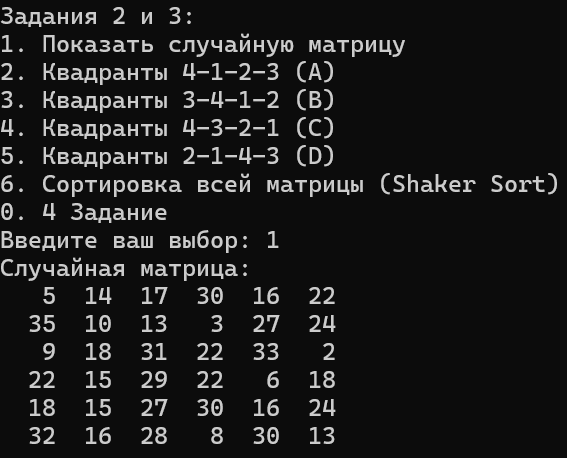
* Shaker sort;
* Comb sort;
* Insert sort;
* Quick sort;

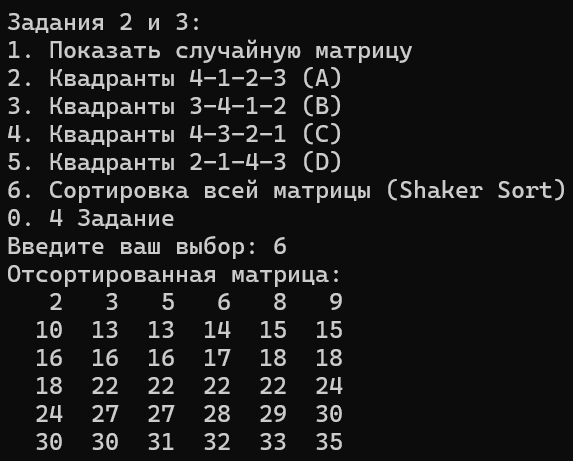
4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число (\*Для манипуляции с элементами используйте только арифметику указателей).

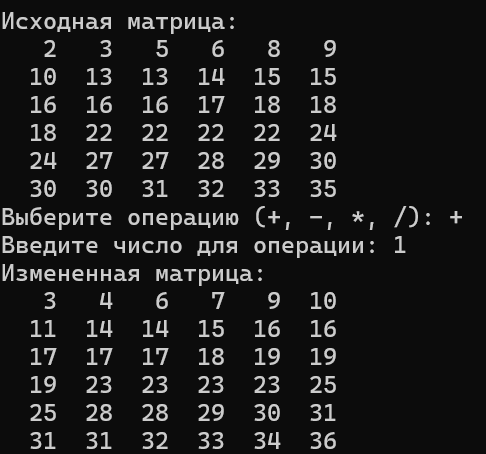
**Выполнение работы.**

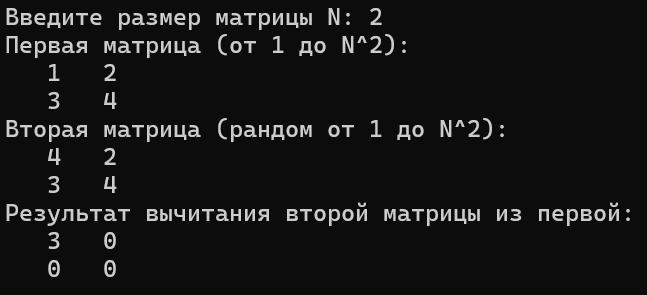
Код программы представлен в приложении А.











**Выводы.**

Я узнал, как работают указатели и ссылки. Научился выводить массив змейкой.

Приложение А

рабочий код

#include <iostream>

#include <chrono>

#include <cstdlib> // Для функции rand() и srand()

#include <ctime>

#include <vector>

using namespace std;

using namespace chrono;

void bubbleSort(int arr[], int n) {

setlocale(LC\_ALL, "RU");

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

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

if (arr[j] > arr[j + 1]) {

swap(arr[j], arr[j + 1]);

}

}

}

cout << "\n";

}

void findMinMax(int arr[], int n, int& min, int& max) {

min = arr[0];

max = arr[0];

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

if (arr[i] < min) {

min = arr[i];

}

if (arr[i] > max) {

max = arr[i];

}

}

cout << "\n";

}

void findIndices(int arr[], int n, int value, vector<int>& indices) {

indices.clear(); // Очистка вектора перед использованием

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

if (arr[i] == value) {

indices.push\_back(i); // Добавление индекса в вектор

}

}

cout << "\n";

}

int countLessThan(int arr[], int n, int a) {

int count = 0;

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

if (arr[i] < a) {

count++;

}

}

return count;

}

int countBiggestThan(int arr[], int n, int b) {

int count = 0;

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

if (arr[i] > b) {

count++;

}

}

return count;

}

bool binarySearch(int arr[], int n, int value) {

int left = 0;

int right = n - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == value)

return true;

else if (arr[mid] < value)

left = mid + 1;

else

right = mid - 1;

}

return false;

}

bool linearSearch(int arr[], int n, int value) {

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

if (arr[i] == value)

return true;

}

return false;

}

void swapElements(int arr[], int index1, int index2) {

swap(arr[index1], arr[index2]);

}

int main() {

setlocale(LC\_ALL, "RU");

const int N = 100;

int arr[N];

srand(static\_cast<unsigned int>(time(0)));

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

arr[i] = rand() % 199 - 99;

}

cout << "Элементы массива до сортировки:" << endl;

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

cout << arr[i] << " ";

}

cout << endl;

cout << "\n";

int value\_to\_decrease;

cout << "Введите значение для уменьшения четных элементов: ";

cin >> value\_to\_decrease;

int end\_val = 9;

int start\_val = 1;

int random\_value = (rand() % (end\_val - start\_val + 1) + start\_val);

int count\_even = 0;

int count\_odd = 0;

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

if (i % 2 == 0) {

arr[i] -= value\_to\_decrease;

if (arr[i] % 2 == 0) {

count\_even += 1;

}

}

else if (i % 2 != 0 && arr[i] % 2 != 0) {

count\_odd += 1;

}

}

cout << "\n";

cout << "Массив после вычитания значения " << value\_to\_decrease << ":" << endl;

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

cout << arr[i] << " ";

}

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

if (i % 2 == 0) {

arr[i] \*= random\_value;

}

}

cout << "\n" << "\n";

cout << "Массив после умножения четных элементов на значение " << random\_value << ":" << endl;

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

cout << arr[i] << " ";

}

cout << "\n" << "\n";

cout << "Количество четных элементов: " << count\_even << endl;

cout << "Количество нечетных элементов: " << count\_odd << endl;

for (int i = 1; i < 10; i++) {

int counter\_del = 0;

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

if (arr[j] % i == 0) {

counter\_del += 1;

}

};

cout << " Кол-во чисел делящихся на " << i << " равно " << counter\_del << endl;

};

//3 zad

// Время поиска минимального и максимального элемента в неотсортированном массиве

auto start = chrono::high\_resolution\_clock::now();

int minUnsorted, maxUnsorted;

findMinMax(arr, N, minUnsorted, maxUnsorted);

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedUnsorted = end - start;

cout << "Минимальный элемент (неотсортированный): " << minUnsorted << endl;

cout << "Максимальный элемент (неотсортированный): " << maxUnsorted << endl;

cout << "Время поиска минимального и максимального элемента (неотсортированный): "

<< elapsedUnsorted.count() << " наносекунд." << endl;

bubbleSort(arr, N);// Сортировка баблсортом

cout << "Элементы массива после сортировки:" << endl;

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

cout << arr[i] << " ";

}

cout << endl;

cout << "\n";

start = chrono::high\_resolution\_clock::now();

int minSorted, maxSorted;

findMinMax(arr, N, minSorted, maxSorted);

end = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedSorted = end - start;

cout << "Минимальный элемент (отсортированный): " << minSorted << endl;

cout << "Максимальный элемент (отсортированный): " << maxSorted << endl;

cout << "Время поиска (отсортированный): " << elapsedSorted.count() << " наносекунд" << endl;

//4 zad

//ср.знач макс и мин элементов

double averageUnsorted = round((minUnsorted + maxUnsorted) / 2.0);

double averageSorted = round((minSorted + maxSorted) / 2.0);

cout << "Среднее значение (неотсортированный): " << averageUnsorted << endl;

cout << "Среднее значение (отсортированный): " << averageSorted << endl;

// Поиск индексов для неотсортированного массива

vector<int> indicesUnsorted;

start = chrono::high\_resolution\_clock::now();

findIndices(arr, N, averageUnsorted, indicesUnsorted);

end = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedIndicesUnsorted = end - start;

cout << "Индексы элементов, равных среднему значению (неотсортированный): ";

for (int index : indicesUnsorted) {

cout << index << " ";

}

cout << "\n";

cout << "Количество: " << indicesUnsorted.size() << endl;

cout << "Время поиска индексов (неотсортированный): " << elapsedIndicesUnsorted.count() << " наносекунд" << endl;

vector<int> indicesSorted;

start = chrono::high\_resolution\_clock::now();

findIndices(arr, N, averageSorted, indicesSorted);

end = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedIndicesSorted = end - start; // Время в наносекундах

cout << "Индексы элементов, равных среднему значению (отсортированный): ";

for (int index : indicesSorted) {

cout << index << " ";

}

cout << "\n";

cout << "Количество: " << indicesSorted.size() << endl;

cout << "Время поиска индексов (отсортированный): " << elapsedIndicesSorted.count() << " наносекунд" << endl;

cout << "\n";

//5 zad

int a;

cout << "Введите число a: ";

cin >> a;

int countLess = countLessThan(arr, N, a);

cout << "Количество элементов в отсортированном массиве, которые меньше числа " << a << ": " << countLess << endl;

cout << "\n";

//6 zad

int b;

cout << "Введите число b: ";

cin >> b;

int countGreater = countBiggestThan(arr, N, b);

cout << "Количество элементов в отсортированном массиве, которые больше числа " << b << ": " << countGreater << endl;

cout << "\n";

//7 zad

int searchValue;

cout << "Введите число для поиска: ";

cin >> searchValue;

// Сравнение скорости бинарного поиска и линейного поиска

auto startLinear = chrono::high\_resolution\_clock::now();

bool foundLinear = linearSearch(arr, N, searchValue);

auto endLinear = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedLinear = endLinear - startLinear; // Время в наносекундах

auto startBinary = chrono::high\_resolution\_clock::now();

bool foundBinary = binarySearch(arr, N, searchValue);

auto endBinary = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedBinary = endBinary - startBinary; // Время в наносекундах

cout << "Результат линейного поиска: " << (foundLinear ? "Найдено" : "Не найдено") << endl;

cout << "Время линейного поиска: " << elapsedLinear.count() << " наносекунд" << endl;

cout << "Результат бинарного поиска: " << (foundBinary ? "Найдено" : "Не найдено") << endl;

cout << "Время бинарного поиска: " << elapsedBinary.count() << " наносекунд" << endl;

cout << "\n";

//8 zad

int index1, index2;

cout << "Введите индексы для обмена местами (от 0 до " << N - 1 << "): ";

cin >> index1 >> index2;

if (index1 >= 0 && index1 < N && index2 >= 0 && index2 < N) {

auto startSwap = chrono::high\_resolution\_clock::now();

swapElements(arr, index1, index2);

auto endSwap = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedSwap = endSwap - startSwap; // Время в наносекундах

cout << "Элементы массива после обмена местами:" << endl;

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

cout << arr[i] << " ";

}

cout << endl;

cout << "\n";

cout << "Время обмена: " << elapsedSwap.count() << " наносекунд" << endl;

}

else {

cout << "Ошибка: индексы должны быть в диапазоне от 0 до " << N - 1 << "." << endl;

}

return 0;

}