**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: ОДНОМЕРНЫЕ СТАТИЧЕСКИЕ МАССИВЫ

|  |  |  |
| --- | --- | --- |
| Студент гр. 3373 |  | Долгий А. М. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Разработать алгоритм и написать программу, которая позволяет работать с одномерными статическими массивами

**Основные теоретические положения.**

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнить однообразные действия, им дают одно имя, а различают по порядковому номеру (индексу). Это дает возможность компактно записать множество операций с использованием циклов.

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.

Все массивы можно разделить на две группы: одномерные и многомерные. Описание массива в программе отличается от объявления обычной переменной наличием размерности массива, которая задается в квадратных скобках после имени.

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных.

Аналогом одномерного массива из математики может служить последовательность некоторых элементов с одним индексом.

​

При i = 0, 1, 2, … n – одномерный вектор. Каждый элемент такой последовательности представляет собой некоторое значение определенного типа данных. Наглядно одномерный массив можно представить как набор пронумерованных ячеек, в каждой из которых содержится определенное значение

**Постановка задачи.**

Необходимо написать программу, которая:

1)    Создает целочисленный массив размерности *N* = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.

2)    Отсортировать заданный в пункте 1 массив […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.

3)    Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.

4)    Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном. Выводит индексы всех элементов, которые равны этому значению, и их количество. Подсчитайте время поиска.

5)    Выводит количество элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем.

6)    Выводит количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.

7)    Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором. (\*)

8)     Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.

Должна присутствовать возможность запуска каждого пункта многократно.

**Выполнение работы.**

Код программы представлен в приложении А.

В первом блоке определяются вспомогательные функции, которые используются далее в программе (Например, функция quickSort(int arr[], int start, int end)).

После этого определяются функции Task1…8(), представляющие собой выполнение заданий 1-8 соответственно.

После этого идёт функция main в которой реализовано меню и возможность повторного выбора задания.

Скриншоты представлены далее в таблице

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| Меню | |
| При запуске программы перед пользователем появляется меню, где он может выбрать нужную задачу. | Меню: |

Продолжение Таблицы

|  |  |
| --- | --- |
| Представление числа в памяти | |
| При вводе пользователем корректного значения пункта меню, пользователь попадает в меню конкретного задания | Как только введено число, программа выводит соответствующее задание: |

|  |  |
| --- | --- |
| Выход из программы | |
| Пользователь может ввести символ “0”, чтобы выйти из программы | Пример ввода 0, который приводит к выходу (в режиме отладки, иначе консоль закрывается сразу): |

**Выводы.**

Я научился работать с одномерными массивами, сортировкой массивов и поиском по одномерным массивам.

Приложение А

рабочий код

#include <random>

#include <iostream>

#include <chrono>

using namespace std;

int MainArray[100]; // для большинства пунктов

int UnsortedArray[100]; // Для пункта 3

//

// Реализация quick sort

//

int partition(int arr[], int start, int end) // Дополнительная функция для quick sort

{

int pivot = arr[start];

int count = 0;

for (int i = start + 1; i <= end; i++) {

if (arr[i] <= pivot)

count++;

}

// Giving pivot element its correct position

int pivotIndex = start + count;

swap(arr[pivotIndex], arr[start]);

// Sorting left and right parts of the pivot element

int i = start, j = end;

while (i < pivotIndex && j > pivotIndex) {

while (arr[i] <= pivot) {

i++;

}

while (arr[j] > pivot) {

j--;

}

if (i < pivotIndex && j > pivotIndex) {

swap(arr[i++], arr[j--]);

}

}

return pivotIndex;

}

void quickSort(int arr[], int start, int end) // Реализация quick sort

{

// Дефолтный случай

if (start >= end)

return;

// Разделение массива

int p = partition(arr, start, end);

// Сортировка левой части

quickSort(arr, start, p - 1);

// Сортировка правой части

quickSort(arr, p + 1, end);

}

int maxMinSearch(int mass[], bool needMin) { // Поиск минимального и максимального для пункта 4

int max = mass[0];

int min = mass[0];

if (needMin) {

for (int i = 1; i < 100; i++)

{

// Checking for min

if (mass[i] < min) {

min = mass[i];

}

}

}

else

{

for (int i = 1; i < 100; i++)

{

// Checking for max

if (mass[i] > max) {

max = mass[i];

}

}

}

if (needMin == false) {

return max;

}

else

{

return min;

}

}

int BinarySearch(int arr[], int key, int l, int r)

{

int m = l + (r - l) / 2;

if (l >= r) return -1;

if (arr[m] == key) return m;

return arr[m] > key ? BinarySearch(arr, key, l, m) : BinarySearch(arr, key, m + 1, r);

}

void task1(){

//

// Пункт 1. Создаём целочисленный массив размерности N = 100. Элементы массива должны принимать случайное значение в диапазоне от -99 до 99.

//

random\_device rd; // non-deterministic generator

mt19937 gen(rd()); // to seed mersenne twister.

uniform\_int\_distribution<> dist(-99, 99); // distribute results between -99 and 99 inclusive.

for (int i = 0; i < 100; i++) {

MainArray[i] = dist(gen);

UnsortedArray[i] = dist(gen);

}

}

void task2() {

//

// Пункт 2. Отсортируем заданный в пункте 1 массив (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.

//

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

quickSort(MainArray, 0, 99);

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

cout << "Elapsed microseconds: " << (chrono::duration\_cast<chrono::microseconds>(ClockStop - ClockStart).count()) << "us\n";

}

int maxx, minn;

void task3() {

//

// Пункт 3. Найдём максимальный и минимальный элемент массива. Подсчитаем время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.

//

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

cout << "Max: " << MainArray[99] << " || Min: " << MainArray[0] << "\n";

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

cout << "Elapsed microseconds (for task 3, sorted array): " << (chrono::duration\_cast<chrono::microseconds>(ClockStop - ClockStart).count()) << "us\n\n";

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

maxx = maxMinSearch(UnsortedArray, false), minn = maxMinSearch(UnsortedArray, true);

cout << "Max: " << maxx << " || Min: " << minn << "\n";

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

cout << "Elapsed microseconds (for task 3, unsorted array): " << (chrono::duration\_cast<chrono::microseconds>(ClockStop - ClockStart).count()) << "us\n";

}

void task4() {

//

// Пункт 4. Выведем среднее значение (если необходимо, число округлим) максимального и минимального значения в отсортированном и неотсортированном. Выведем индексы всех элементов, которые равны этому значению, и их количество

//

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

int avg = (MainArray[99] + MainArray[0]) / 2;

cout << "Average of maximum and minimum values (sorted array): " << avg << "\n";

int k = 0;

for (int i = 0; i < 100; i++)

{

if (MainArray[i] == (avg)) {

printf("%d|", i); k++;

}

} cout << "\nAmount: " << k << "\n";

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

cout << "Elapsed microseconds (for task 4, sorted array): " << (chrono::duration\_cast<chrono::microseconds>(ClockStop - ClockStart).count()) << "us\n\n";

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

maxx = maxMinSearch(UnsortedArray, false), minn = maxMinSearch(UnsortedArray, true);

cout << "Average of maximum and minimum values (unsorted array): " << (maxx + minn) / 2 << "\n";

k = 0;

for (int i = 0; i < 100; i++)

{

if (UnsortedArray[i] == (maxx + minn) / 2) {

printf("%d|", i); k++;

}

} cout << "\nAmount: " << k << "\n";

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

cout << "Elapsed microseconds (for task 4, unsorted array): " << (chrono::duration\_cast<chrono::microseconds>(ClockStop - ClockStart).count()) << "us\n";

}

void task5() {

//

// Пункт 5. Выводем количество элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем.

//

int a;

cout << "Enter a number to show the quantity of integers in the array that are less than the specified one:\n";

cin >> a;

int k = 0;

for (int i = 0; i < 100; i++) {

if (MainArray[i] < a) {

k++;

}

else { break; }

}

cout << "\nQuantity: " << k;

}

void task6() {

//

// Пункт 6. Выводем количество элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем.

//

int a;

cout << "Enter a number to show the quantity of integers in the array that are bigger than the specified one:\n";

cin >> a;

int k = 0;

for (int i = 0; i < 100; i++) {

if (MainArray[i] > a) k++;

}

cout << "\nQuantity: " << k;

}

int permanentc;

void task7() {

//

// Пункт 7(\*). Выведем информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуем алгоритм бинарного поиска. Сравним скорость его работы с обычным перебором.

//

int c;

cout << "Enter a number to get it's index in a sorted array:\n";

cin >> c;

permanentc = c;

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

c = BinarySearch(MainArray, c, 0, 99);

if (c == -1) { cout << "[BS]Specified number doesnt present in the array"; } // BS AKA binary search, BF AKA bruteforce

else { cout << "[BS]Specified number does present in the array. Index of that number is: " << c; }

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

cout << "\nElapsed microseconds (for task 7, binary search): " << (chrono::duration\_cast<chrono::microseconds>(ClockStop - ClockStart).count()) << "us\n";

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

int i;

for (i = 0; i < 100; i++) {

if (MainArray[i] == permanentc) {

cout << "\n[BF]Specified number does present in the array. Index of that number is: " << i; i=102;

}

}

if (i == 100) {

cout << "\n[BF]Specified number doesnt present in the array";}

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

cout << "\nElapsed microseconds (for task 7, bruteforce): " << (chrono::duration\_cast<chrono::microseconds>(ClockStop - ClockStart).count()) << "us\n";

}

void task8(){

//

// Пункт 8. Поменяем местами элементы массива, индексы которых вводит пользователь. Выведем скорость обмена, используя библиотеку chrono.

//

int x, y;

cout << "Enter the indexes of the numbers X and Y:\n";

cin >> x >> y;

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

cout << "X = " << MainArray[x] << " Y = " << MainArray[y] << "\nChanging places...\n";

int buf;

buf = MainArray[x];

MainArray[x] = MainArray[y];

MainArray[y] = buf;

cout << "X = " << MainArray[x] << " Y = " << MainArray[y] << "\n";

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

cout << "\nElapsed microseconds: " << (chrono::duration\_cast<chrono::microseconds>(ClockStop - ClockStart).count()) << "us\n";

}

int main() {/\*

task1();

cout << "---Task 2---\n\n";

task2();

cout << "\n\n---Task 3---\n\n";

task3();

cout << "\n\n---Task 4---\n\n";

task4();

cout << "\n\n---Task 5---\n\n";

task5();

cout << "\n\n---Task 6---\n\n";

task6();

cout << "\n\n---Task 7---\n\n";

task7();

cout << "\n\n---Task 8---\n\n";

task8();\*/

int MenuPick;

menupick:

cout << "\n\n1. Task 1\n2. Task 2\n3. Task 3\n4. Task 4\n5. Task 5\n6. Task 6\n7. Task 7\n8. Task 8\n0. Exit\n\n";

cin >> MenuPick;

switch (MenuPick) {

case(1):

system("cls"); task1(); system("cls"); cout << "Task 1 done. Going back to menu.\n"; goto menupick; break;

case(2):

system("cls"); cout << "---Task 2---\n\n"; task2();

goto menupick; break;

case(3):

system("cls"); cout << "---Task 3---\n\n"; task3();

goto menupick; break;

case(4):

system("cls"); cout << "---Task 4---\n\n"; task4();

goto menupick; break;

case(5):

system("cls"); cout << "---Task 5---\n\n"; task5();

goto menupick; break;

case(6):

system("cls"); cout << "---Task 6---\n\n"; task6();

goto menupick; break;

case(7):

system("cls"); cout << "---Task 7---\n\n"; task7();

goto menupick; break;

case(8):

system("cls"); cout << "---Task 8---\n\n"; task8();

goto menupick; break;

case(0):

system("cls"); exit(0); break;

}

}