**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. 0324 |  | Наквасин А.А. |
| Преподаватель |  | Глущенко А.Г |

Санкт-Петербург

2020

**Цель работы.**

Знакомство с массивами и способами работы с ними, а также исследование различных способов сортировки элементов массивов.

**Основные теоретические положения.**

**Массивы.**

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: *ai*​ 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.

Объявление в программах одномерных массивов выполняется в соответствии со следующим правилом:

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

Значения индексов элементов массивов всегда начинается с 0. Поэтому максимальное значение индекса элемента в массиве всегда на единицу меньше количества элементов в массиве.

Обращение к определенному элементу массива осуществляется с помощью указания значения индекса этого элемента: А[8]. 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. Например, поскольку базовым типом массива A является тип данных int, с любым элементом этого массива можно выполнять любые операции, которые можно выполнять над значениями типа int.

Значения всех элементов массива в памяти располагаются в непрерывной области одно за другим. Общий объем памяти, выделяемый компилятором для массива, определяется как произведение объема одного элемента массива на количество элементов в массиве и равно:

sizeof( <Базовый тип> )  \* <Количество элементов>

**Виды сортировок.**

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Когда элементы отсортированы, их проще найти, производить с ними различные операции. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива.

Простейшая из сортировок – сортировка обменом (**пузырьковая сортировка**). Вся суть метода заключается в попарном сравнении элементов и последующем обмене. Таким образом, если следующий элемент меньше текущего, то они меняются местами, максимальный элемент массива постепенно смещается в конец массива, а минимальный – в начало. Один полный проход по массиву может гарантировать, что в конце массива находится максимальный элемент.

Затем процесс повторяется до тех пор, пока вся последовательность не будет упорядочена. Важно заметить, что после первого прохода по массиву, уже имеется один упорядоченный элемент, он стоит на своем месте, и менять его не надо. Таким образом на следующем шаге будут сравниваться N-1 элемент.

Очевидно, что хуже всего алгоритм будет работать, когда на вход подается массив, отсортированный в обратную сторону (от большего к меньшу). Быстрее же всего алгоритм работает с уже отсортированным массивом.

Но стандартный алгоритм пузырьковой сортировки предполагает полный циклический проход по массиву. Если изначально подается упорядоченная последовательность, то работа алгоритма все равно продолжиться. Исправить это можно, добавив условие проверки: если на текущей итерации ни один элемент не изменил свой индекс, то работа алгоритма прекращается.

**Shaker sort** – модификация пузырьковой сортировки. Принцип работы этой сортировки аналогичен bubble sort: попарное сравнение элементов и последующий обмен местами. Но имеется существенное отличие. Как только максимальный элемент становится на свое место, алгоритм не начинает новую итерацию с первого элемента, а запускает сортировку в обратную сторону. Алгоритм гарантирует, что после выполнения первой итерации, минимальный и максимальный элемент будут в начале и конце массива соответственно.

Затем процесс повторяется до тех пор, пока массив не будет отсортирован. За счет того, что сортировка работает в обе стороны, массив сортируется на порядок быстрее. Очевидным примером этого был бы случай, когда в начале массива стоит максимальный элемент, а в конце массива – минимальный. Shaker sort справится с этим за 1 итерацию, при условии, что другие элементы стоят на правильном месте.

Кажется, что bubble sort теряет свою эффективность по сравнению с shaker sort. Сортировка проходит в массиве в обоих направлениях, а не только от его начала к концу. Но в работе с большими массивами преимущество шейкер-сортировки уменьшается как раз из-за использования двух циклов.

**Сортировка вставками (insert sort)** – алгоритм сортировки, в котором элементы массива просматриваются по одному, и каждый новый элемент размещается в подходящее место среди ранее упорядоченных элементов.

Общая суть сортировки вставками такова:

1)    Перебираются элементы в неотсортированной части массива.

2)    Каждый элемент вставляется в отсортированную часть массива на то место, где он должен находится.

Сортировка вставками делить массив на 2 части – отсортированную и неотсортированную. С каждым новым элементом отсортированная часть будет увеличиваться, а неотсортированная уменьшаться. Причем найти нужное место для очередного элемента в отсортированном массиве достаточно легко.

Рассмотрим самый простой способ (рис. 3.5). Необходимо пройти массив слева направо и обработать каждый элемент. Слева будет наращиваться отсортированная часть массива, а справа – уменьшаться неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента. Сам элемент отправляется в буфер, что освобождает место в массиве и позволяет сдвинуть элементы и освободить точку вставки.

Существует множество модификаций сортировки вставками, некоторые из них затрагивают именно способ вставки элемента в отсортированную часть. Одна из самых лучших модификаций – сортировка простыми вставками с бинарным поиском. Бинарный поиск будет описан позже.

Лучше всего сортировка вставками работает при обработке почти отсортированных массивов. В таком случае insert sort работает быстрее других сортировок.

**Быстрая сортировка (quick sort)** – одна из самых быстрых сортировок. Эта сортировка по сути является существенно улучшенной версией алгоритма пузырьковой сортировки.

Общая идея алгоритма состоит в том, что сначала выбирается из массива элемент, который называется опорным. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Затем необходимо сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: меньше опорного, раны опорному и больше опорного. Для меньших и больших значений необходимо выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

**Постановка задачи.**

Нужно разработать алгоритм и написать программу, которая позволяет:

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 элементы массива […] 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. Выводит индексы всех элементов, которые равны этому значению, и их количество.

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.**

Для решения поставленных была создана программа на языке программирования C++.

В первой задаче был использован цикл for, который заполнял созданный пустой массив случайными числами при помощи функции srand.

Во второй задаче для сортировки массива использовался метод Quick sort.

В третьей задаче для поиска максимального и минимального элементов в неотсортированном массиве использовался метод перебора всех элементов массива, а в отсортированном вывод первого и последнего элементов массива .

В четвертой задаче производится сложение первого и последнего элементов отсортированного массива и деление их суммы на два. После чего с помощью перебора ищутся идентичные элементы массива.

В пятой задаче перебираются все элементы отсортированного массива и подсчитывается сколько из них меньше числа, введённого пользователем.

В шестой задаче перебираются все элементы отсортированного массива и подсчитывается сколько из них больше числа, введённого пользователем.

В седьмой задаче реализуется алгоритм бинарного поиска и сравнивается его скорость работы со скорость простого перебора всех элементов.

В восьмой задаче мы запрашиваем у пользователя индексы элементов массива которые требуется поменять местами и используем для этого функцию swap.

**Итоговый код программы:**

#include <iostream>

#include <chrono>

#include <ctime>

int binary\_search(int \*O, int a)

{

int start = 0, end = 99;

int middle;

bool check = false;

do

{

middle = (start + end) / 2;

if (O[middle] == a)

{

std::cout << "\nЧисло найденно";

check = true;

break;

}

else if (O[middle] > a)

end = middle;

else if (O[middle] < a)

start = middle;

} while (start < end);

if(check == false)

std::cout << "\nЧисло не найденно";

return 0;

}

int max\_min(int \*N)

{

int max = -99, min = 99;

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

{

if (N[i] > max)

max = N[i];

if (N[i] < min)

min = N[i];

}

std::cout << "\n\nМинимальный элемент: " << min << "\nМаксимальный элемент: " << max;

return 0;

}

void quick\_sort(int \*O, int end, int begin)

{

int mid, a;

int f = begin;

int l = end;

mid = O[(f + l) / 2];

while (f < l)

{

while (O[f] < mid) f++;

while (O[l] > mid) l--;

if (f <= l)

{

a = O[f];

O[f] = O[l];

O[l] = a;

f++;

l--;

}

}

if (begin < l) quick\_sort(O, l, begin);

if (f < end) quick\_sort(O, end, f);

}

int one(int \*N)

{

srand(time(0));

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

{

N[i] = rand() % 198 - 99;

std::cout << N[i] << " ";

}

return 0;

}

int two(int \*N, int \*O)

{

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

O[i] = N[i];

auto start = std::chrono::steady\_clock::now();

quick\_sort(O, 99, 0);

auto end = std::chrono::steady\_clock::now();

auto time = std::chrono::duration<double>(end - start);

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

std::cout << O[i] << " ";

std::cout << "\n\nВремя сортировки массива методом быстрой сортировкой: " << std::fixed << time.count() << " секунд";

return 0;

}

int three(int \*N, int \*O)

{

auto start = std::chrono::steady\_clock::now();

max\_min(N);

auto end = std::chrono::steady\_clock::now();

auto time = std::chrono::duration<double>(end - start);

std::cout << "\nВремя поиска в неотсортированном массиве: " << std::fixed << time.count() << " секунд";

start = std::chrono::steady\_clock::now();

int max = O[99];

int min = O[0];

end = std::chrono::steady\_clock::now();

time = std::chrono::duration<double>(end - start);

std::cout << "\n\nМинимальный элемент: " << min << "\nМаксимальный элемент: " << max << "\nВремя поиска в отсортированном массиве: " << std::fixed << time.count() << " секунд";

return 0;

}

int four(int \*O)

{

int middle = (O[0]+O[99])/2;

int count = 0;

std::cout << "\nСреднее значение элементов массива(max/min): " << middle;

for (int i = 0; i < 99; i++)

{

if (O[i] == middle)

{

count++;

std::cout << "\nИндекс элемента со значением равным среднему: " << i;

}

}

std::cout << "\nКоличество совпадений: " << count;

return 0;

}

int five(int \*O)

{

int a;

int count = 0;

std::cout << "\nВведите число для сравнения: ";

std::cin >> a;

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

{

if (O[i] < a)

count++;

}

std::cout << "\nКоличество элементов массива менших заданного числа: " << count;

return 0;

}

int six(int \*O)

{

int b;

int count = 0;

std::cout << "\nВведите число для сравнения: ";

std::cin >> b;

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

{

if (O[i] > b)

count++;

}

std::cout << "\nКоличество элементов массива больших заданного числа: " << count;

return 0;

}

int seven(int \*O)

{

int a;

std::cout << "\nВведите искомое число: ";

std::cin >> a;

auto start = std::chrono::steady\_clock::now();

binary\_search(O, a);

auto end = std::chrono::steady\_clock::now();

auto time = std::chrono::duration<double>(end - start);

std::cout << "\nВремя потраченное на поиск бинарным методом: " << std::fixed << time.count();

start = std::chrono::steady\_clock::now();

for (int i = 0; i < 99; i++)

{

if (O[i] == a)

{

std::cout << "\nЧисло найденно";

break;

}

if (O[i] > a || i>97)

{

std::cout << "\nЧисло не найденно";

break;

}

}

end = std::chrono::steady\_clock::now();

time = std::chrono::duration<double>(end - start);

std::cout << "\nВремя потраченное на поиск перебором: " << std::fixed << time.count();

return 0;

}

int eight(int\* O)

{

int a, b, swap;

std::cout << "\nВведите индекс первого числа для обмена: ";

std::cin >> a;

std::cout << "\nВведите индекс второго числа для обмена: ";

std::cin >> b;

auto start = std::chrono::steady\_clock::now();

swap = O[a];

O[a] = O[b];

O[b] = swap;

auto end = std::chrono::steady\_clock::now();

auto time = std::chrono::duration<double>(end - start);

std::cout << "\nВремя потраченное на обмен: " << std::fixed << time.count();

return 0;

}

int main()

{

int N[100];

int O[100];

int i;

setlocale(0, "");

std::cout << "1 - Создание массива случайных чисел\n2 - Сортировка массива\n3 - Поиск минимального и максимального элементов\n4 - Среднее значение\n\

5 - Поиск значений меньших заданного\n6 - Поиск значений больших заданного\n7 - Поиск заданного числа\n8 - Обмен местами двух элементов\n9 - Выход";

menu: std::cout << "\n\nВыберите пункт меню: ";

std::cin >> i;

switch (i)

{

case 1:

std::cout << "\nМассив случайных чисел: ";

one(N);

goto menu;

case 2:

std::cout << "\nОтсортированный массив случайных чисел: ";

two(N, O);

goto menu;

case 3:

three(N, O);

goto menu;

case 4:

four(O);

goto menu;

case 5:

five(O);

goto menu;

case 6:

six(O);

goto menu;

case 7:

seven(O);

goto menu;

case 8:

eight(O);

goto menu;

case 9:

break;

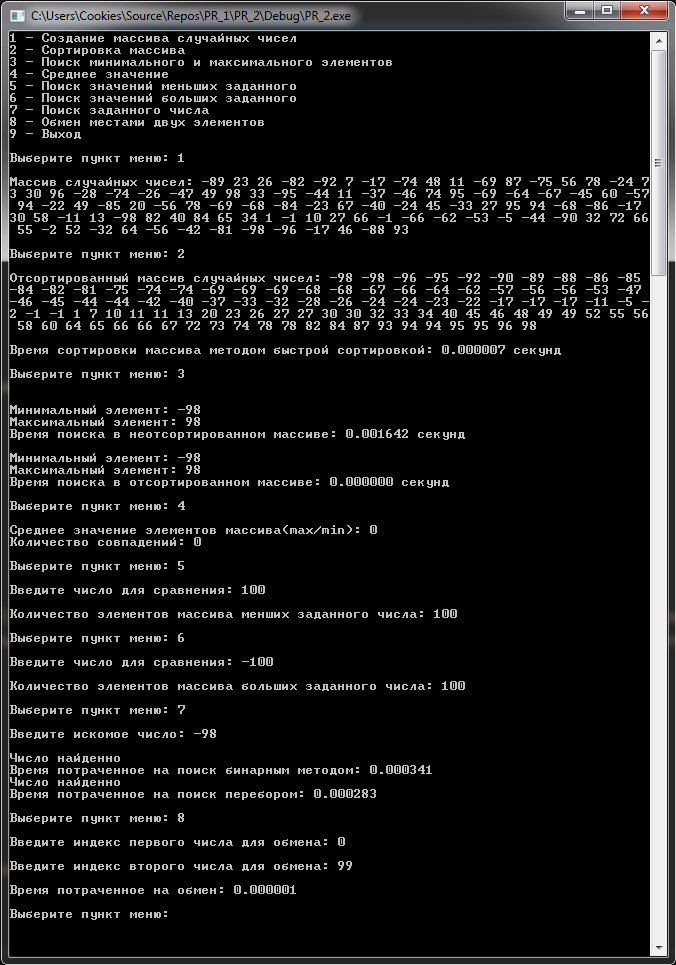
default:

std::cout << "\nТакого пункта не существует\n";

goto menu;

}

**Результат работы программы.**

****

**Вывод.**

При разработке программы мною были изучены различные виды сортировок массива, такие как: пузырьковая сортировка, шейкер-сортировка, сортировка расчёской, сортировка вставками и быстрая сортировка. Также мной был метод изучен метод бинарного поиска, позволяющий значительно ускорить поиск необходимого элемента в отсортированном массиве с большим количеством элементов. Библиотека chrono позволяет замерять быстродействие отдельных функций и участков кода, что позволяет выбирать более рациональное решение поставленной проблемы .