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

Студентка гр. 0324 Серебрякова А.К.

Преподаватель Глущенко А.Г.

Санкт-Петербург

2020

Цель работы.

Изучение структуры одномерных массивов, обработка данных одномерных массивов. Изучение различных видов сортировок. Проведение временной оценки различных действий с массивами.

Основные теоретические положения.

Понятие массива.

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. В листинге 3.1 представлен пример работы с одномерным массивом. Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных. Инициализировать массив можно и другим более простым способом: инициализирующие значения записываются в фигурных скобках. Значения элементам присваиваются по порядку. Если элементов в массиве больше, чем инициализаторов, элементы, для которых значения не указаны, обнуляются:

int arr[4] = {3, 2, 1}; // arr[0] = 3, arr[1] = 2, arr[2] = 1, arr[3] = 0

Размерность массива вместе с типом его элементов определяет объем памяти, необходимый для размещения массива, которое выполняется на этапе компиляции, поэтому размерность должна быть задана целой положительной константой или константным выражением.

Пример заполнения одномерного массива

/\* arr.cpp: Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы \*/

#include

#include

using namespace std;

int main()

{ setlocale(LC\_ALL, "Russian");

const int N = 10; // Задаём размерность массива

srand(time(0));

int arr[N]; // Объявляем массив arr размерности N

cout << "Исходный массив данных: ";

for (int i = 0; i < N; i++) // Цикл, заполняющий массив случайными числами

{ arr[i] = rand() % 9; // Заполняем массив случайными числами от 0 до 9

cout << arr[i] << " "; // Выводим элемент массива на экран

}

return 0;

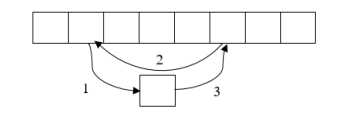
}

Если при описании массива не указана размерность, массив обязательно должен быть инициализирован. Компилятор сам определит размерность массива по количеству элементов:

int arr[] = {7, 6, 5, 4, 3, 2, 1}; // Всего 7 элементов

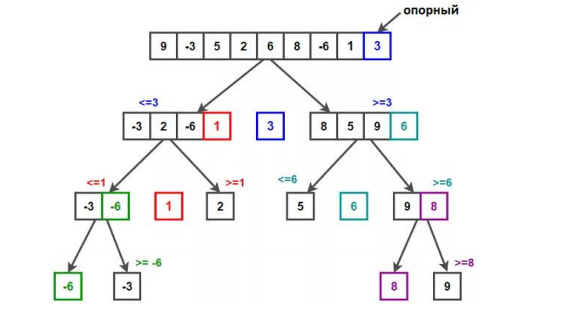
Для доступа к элементу массива после его имени указывается номер элемента (индекс) в квадратных скобках. Важно помнить, что при обращении к элементам массива автоматический контроль выхода индекса за границу массива не производится, это может привести к ошибкам. Обмен элементов массива осуществляется через буферную переменную либо через функцию swap(a, b)

Обмен местами элементов массива:



Определить, сколько памяти выделено под массив, можно с помощью операции sizeof. Необходимо определить, сколько выделяется памяти на 1 элемент массива и умножить на количество элементов массива. Принцип действия схож с задачей про переливание воды из одного сосуда в другой. Чтобы поменять жидкость сосудов местами, нужно использовать еще один сосуд, который будет временно содержать жидкость одного из сосудов.

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.



На практике массив обычно делят на две части: «меньше опорного» и «равные и большие» или «меньше опорного или равные» и «большие». Такой поход в общем случае эффективнее, ведь упрощается алгоритм разделения. При том, что это один из самых быстродействующих из алгоритмов, данные алгоритм сортировки неустойчив, а прямая реализация в виде функции с двумя рекурсивными вызовами может привести к ошибке переполнения стека.

Алгоритм бинарного поиска

Алгоритм бинарного поиска – классический алгоритм поиска в отсортированном массиве, который использует дробление массива на половины. Если элемент, который необходимо найти, присутствует в списке, то бинарный поиск возвращает ту позицию, в которой он был найден. Рассмотрим простой пример: имеется массив из 100 элементов, упорядоченных по возрастанию от 1 до 100. Было загадано какое-то число, необходимо его назвать. Компьютер имеет три ответа на ваше предположение: верно, число больше, число меньше. Сколько попыток нужно, чтобы ответить правильно? Обычный перебор – наихудшая стратегия. Можно назвать правильный ответ лишь с 100-ой попытки. Но если начать спрашивать с середины, то ситуация кардинально меняется. Если число больше 50, то необходимо делить правую половину, и следующее предположение – 75, если меньше – 25. Так необходимо продолжать до тех пор, пока не будет названо правильное число. Наибольшее число предположений равняется:



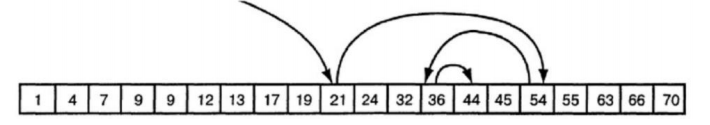
Этот результат явно лучше простого перебора. Схожий принцип работы имеет алгоритм бинарного поиска. Бинарный поиск работает только в топ случае, если список отсортирован. Например, если бы искомое минимальное значение стояло не на своем положенном месте, а на месте максимального элемента, то мы бы откинули его на первой же итерации. Сам алгоритм имеет вид:

1) Определение значения в середине массива (или иной структуры данных). Полученное значение сравнивается с ключом (значением, которое необходимо найти).

2) Если ключ меньше значения середины, то необходимо осуществлять поиск в первой половине элементов, иначе – во второй.

3) Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.

4) Процесс продолжается до тех пор, пока не будет определен элемент, равный значению ключа или не станет пустым интервал для поиска.



Бинарный поиск числа 44

Чтобы уменьшить количество шагов поиска, можно сразу смещать границы поиска на элемент, следующий за серединой отрезка.

Постановка задачи.

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.

Выполнение работы.

1. Чтобы заполнить массив случайными числами, использую rand() – генератор случайных чисел

2. Заданный в пункте 1 массив сортирую при помощи быстрой сортировки (quick sort)

3. Сначала нахожу в неотсортированном массиве максимум и минимум, замеряю время поиска, затем в отсортированном массиве, также замеряя время поиска

4. Через цикл for нахожу среднее значение, которое было подсчитано ранее.

5. Нахожу в массиве число, которое меньше введенного, записываю его индекс в новую переменную. Через цикл for вывожу элементы от найденного индекса в обратном порядке

6. Нахожу в массиве число, которое больше введенного, записываю его индекс в новую переменную. Через цикл for вывожу элементы от найденного индекса.

7. Алгоритм бинарного поиска:

1)    Определение значения в середине массива (или иной структуры данных). Полученное значение сравнивается с ключом (значением, которое необходимо найти).

2)    Если ключ меньше значения середины, то необходимо осуществлять поиск в первой половине элементов, иначе – во второй.

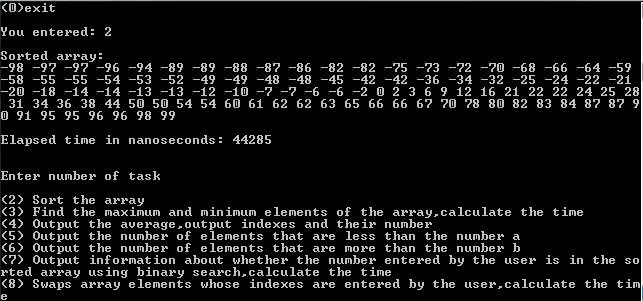
3)    Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.

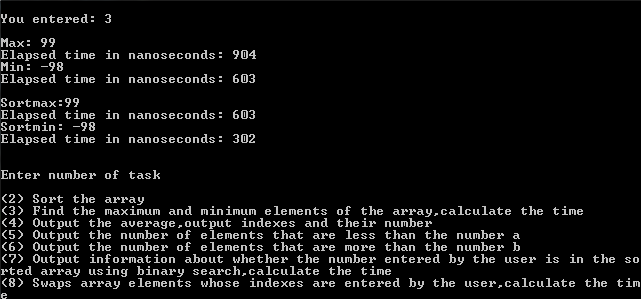
4)    Процесс продолжается до тех пор, пока не будет определен элемент, равный значению ключа или не станет пустым интервал для поиска.

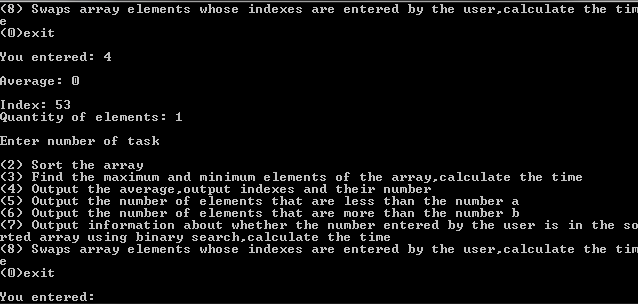
Замеряю затраченное время. После записываю перебор, также замеряю время.

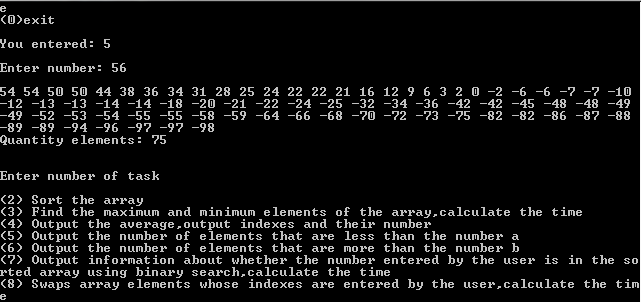
8. При помощи буферной переменной меняю элементы массива, после массив выводится на экран.

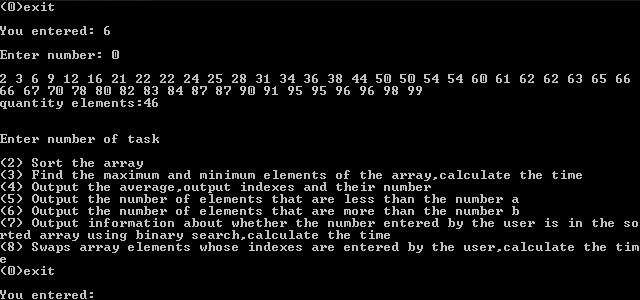
Примеры работы

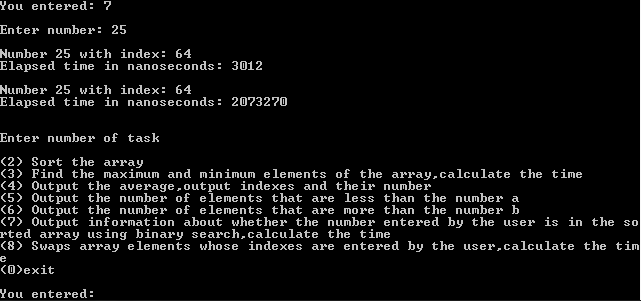


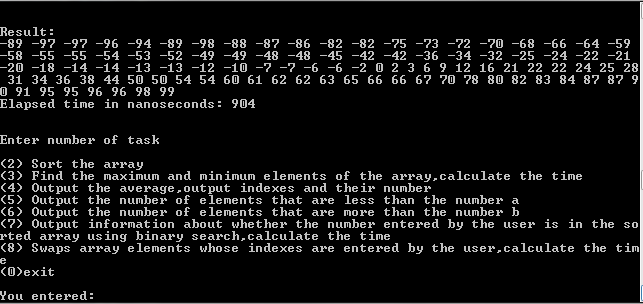












Код программы.

#include <iostream>

#include <chrono>

using namespace std;

using namespace std::chrono;

void quicksort(int \*mas, int endmas, int begin)

{

int mid; // середина

int left = begin; // левая граница массива,0

int right = endmas; // правая граница массива,n - 1

mid = mas[(left + right) / 2];

while (left < right)

{

while (mas[left] < mid) left++; // двигаем границу массива к концу,пока не найдется элемент больше опорного

while (mas[right] > mid) right--; // двигаем границу массива к началу,пока не найдется элемент меньше опорного

if (left <= right)

{

swap(mas[left], mas[right]); // обмен найдеными элементами

left++; // для проверки следующих элементов

right--;

}

}

if (begin < right) quicksort(mas, right, begin); // продолжение сортировки элементов слева от опорного

if (left < endmas) quicksort(mas, endmas, left); // продолжение сортировки элементов справа от опорного

}

int main()

{

int const n = 100;

int endmas = n - 1, begin = 0;

int i, j;

int mas[n];

int max, min, maxsort, minsort;

int buff = 0;

int count = 0, counting\_elements = 0, counting\_elements2 = 0;

int average;

int a, ind, b, ind2;

int number, left, right, check, mid, find = 0;

int temp, swap1, swap2;

int task;

// заполнение массива для заданий

cout << endl << "Not sorted array: " << "\n";

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

{

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

cout << mas[i] << " ";

}

cout << "\n";

cout << "\n";

// поиск максимума в неотсортированном массиве

auto startmax = steady\_clock::now();

max = -100;

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

{

if (mas[i] > max)

max = mas[i];

}

auto endmax = steady\_clock::now();

auto elepsed\_max = duration\_cast<nanoseconds>(endmax - startmax);

//поиск минимума в неотсортированном массиве

auto startmin = steady\_clock::now();

min = 100;

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

{

if (mas[i] < min)

min = mas[i];

}

auto endmin = steady\_clock::now();

auto elepsed\_min = duration\_cast<nanoseconds>(endmin - startmin);

// сортировка Quick sort

auto startsort = steady\_clock::now();

quicksort(mas, endmas, begin);

auto endsort = steady\_clock::now();

auto elepsed\_sort = duration\_cast<nanoseconds>(endsort - startsort);

// поиск максимума в отсортированном массиве

auto startmaxsort = steady\_clock::now();

maxsort = -100;

for (int i = 99; i > 0; i--) // проверка значений с 99 индекса в обратном направлении

{

if (mas[i] > maxsort)

maxsort = mas[i];

}

auto endmaxsort = steady\_clock::now();

auto elepsed\_maxsort = duration\_cast<nanoseconds>(endmaxsort - startmaxsort);

// поиск минимума в отсортированном массиве

auto startminsort = steady\_clock::now();

minsort = 100;

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

{

if (mas[i] < minsort)

minsort = mas[i];

}

auto endminsort = steady\_clock::now();

auto elepsed\_minsort = duration\_cast<nanoseconds>(endminsort - startminsort);

do

{

cout << "\n\n";

cout << "Enter number of task\n " << endl;

cout << "(2) Sort the array" << endl;

cout << "(3) Find the maximum and minimum elements of the array,calculate the time" << endl;

cout << "(4) Output the average,output indexes and their number" << endl;

cout << "(5) Output the number of elements that are less than the number a" << endl;

cout << "(6) Output the number of elements that are more than the number b" << endl;

cout << "(7) Output information about whether the number entered by the user is in the sorted array using binary search" << endl;

cout << "(8) Swaps array elements whose indexes are entered by the user" << endl;

cout << "(0)exit" << endl;

cout << "\n";

cout << "You entered: ";

cin >> task;

switch (task)

{

case 2:

cout << "\n" << "Sorted array: " << "\n";

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

cout << mas[i] << ' ';

cout << "\n\n";

cout << "Elapsed time in nanoseconds: " << elepsed\_sort.count() << "\n";

break;

case 3:

cout << "\n";

cout << "Max: " << max << " \n";

cout << "Elapsed time in nanoseconds: " << elepsed\_max.count() << "\n";

cout << "Min: " << min << " \n";

cout << "Elapsed time in nanoseconds: " << elepsed\_min.count() << "\n";

cout << "\n";

cout << "Sortmax:" << maxsort << " \n";

cout << "Elapsed time in nanoseconds: " << elepsed\_maxsort.count() << "\n";

cout << "Sortmin: " << minsort << " \n";

cout << "Elapsed time in nanoseconds: " << elepsed\_minsort.count() << "\n";

break;

case 4:

// подсчет среднего значения

average = min + max;

average = average / 2;

cout << "\n";

cout << "Average: " << average << "\n";

cout << "\n";

//вывод индексов и их количества

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

{

if (mas[i] == average)

{

cout << "Index: " << i << "\n";

count++;

}

}

cout << "Quantity of elements: " << count;

break;

case 5:

//Вывод количества элементов в отсортированном массиве, которые меньше числa a

cout << "\n";

cout << "Enter number: ";

cin >> a;

cout << "\n";

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

{

if (mas[i] < a) // если число в массиве меньше,то

{

ind = i; // в ind записывается индес числа,чтобы понять откуда считать

}

}

for (i = ind; i >= 0; i--)

{

cout << mas[i] << " ";

counting\_elements++; // счетчик для количества элементов массива

}

cout << "\nQuantity elements: " << counting\_elements << "\n";

counting\_elements = 0;

break;

case 6:

//Вывод количества элементов в отсортированном массиве, которые больше числа b

cout << "\n";

cout << "Enter number: ";

cin >> b;

cout << "\n";

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

{

if (mas[i] > b)

{

ind2 = i;

break;

}

}

for (int i = ind2; i < n; i++)

{

cout << mas[i] << " ";

counting\_elements2++;

}

cout << "\nquantity elements:" << counting\_elements2 << "\n";;

counting\_elements2 = 0;

break;

case 7:

if (task == 7)

{

// бинарный поиск

cout << endl << "Enter number: ";

cin >> number;// число,которое нужно найти

left = 0; // левая граница

right = n - 1; // правая граница

check = 0;

cout << "\n";

auto startbin = steady\_clock::now();

while ((left <= right) && (check != 1))

{

mid = (left + right) / 2; // срединный индекс

if (mas[mid] == number) //проверка серединного элемента

{

check++;

break;

}

if (mas[mid] > number) right = mid - 1; // какую часть нужно отбросить

else left = mid + 1;

}

auto endbin = steady\_clock::now();

if (check == 1)

cout << "Number " << number << " " << "with index: " << mid;

else cout << "No entered number";

auto elepsed\_bin = duration\_cast<nanoseconds>(endbin - startbin);

cout << "\n";

cout << "Elapsed time in nanoseconds: " << elepsed\_bin.count() << "\n\n";

// перебор массива

auto startenumeration = steady\_clock::now();

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

{

if (mas[i] == number)

{

cout << "Number " << number << " " << "with index: " << i;

find++;

}

}

auto endenumeration = steady\_clock::now();

if (find == 0)

cout << "No entered number";

cout << "\n";

auto elepsed\_enumeration = duration\_cast<nanoseconds>(endenumeration - startenumeration);

cout << "Elapsed time in nanoseconds: " << elepsed\_enumeration.count() << "\n";

}

break;

case 8:

if (task == 8)

{

cout << "\nEnter first index of the array:";

cin >> swap1;

cout << "\nEnter second index of the array:";

cin >> swap2;

cout << "\n";

// обмен значениями через буферную переменную

auto startsw = steady\_clock::now();

temp = mas[swap1];

mas[swap1] = mas[swap2];

mas[swap2] = temp;

auto endsw = steady\_clock::now();

cout << "\n";

cout << "Result:" << "\n";

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

{

cout << mas[i] << " ";

}

auto elepsed\_sw = duration\_cast<nanoseconds>(endsw - startsw);

cout << "\nElapsed time in nanoseconds: " << elepsed\_sw.count() << "\n";

}

break;

case 0:

break;

}

} while (task != 0);

return 0;

}