**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. 3372 |  | Поконечный А.О |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Изучение и организация структур; получение практических навыков работы со структурами; определение преимуществ и недостатков использования структур.

**Основные теоретические положения.**

Структуры представляют собой группы связанных между собой, как правило, разнотипных переменных, объединенных в единый объект, в отличие от массива, все элементы которого однотипны. В языке C++ структура является видом класса и обладает всеми его свойствами. Чаще всего ограничиваются тем, как структуры представлены в языке С:

struct [имя\_типа] {

тип\_1 элемент\_1;

тип \_2 элемент\_2;

…

тип\_k элемент\_k;

} [ список\_описателей ];

Описание структуры начинается ключевым словом struct. Каждая входящая в структуру переменная называется членом (полем, элементом) структуры и описывается типом данных и именем. Поля структуры могут быть любого типа данных. Их количество не лимитировано.

Вся эта конструкция является инструкцией языка программирования, поэтому после нее всегда должен ставиться символ ‘;’.

При описании структуры память для размещения данных не выделяется. Работать с описанной структурой можно только после того, как будет определена переменная (переменные) этого типа данных, только при этом компилятор выделит необходимую память.

Для инициализации структуры значения ее элементов перечисляют в фигурных скобках в порядке их описания:

struct complex{

float real, im;

} data [2][2] = {

{{1,1}, {2,2}},

{{3,3}, {4,4}}

};

Все поля структурных переменных располагаются в непрерывной области памяти одно за другим. Общий объем памяти, занимаемый структурой, равен сумме размеров всех полей структуры. Для определения размера структуры следует использовать инструкцию sizeof().

Для того чтобы записать данные в структурную переменную, необходимо каждому полю структуры присвоить определенное значение. Для этого необходимо использовать оператор ‘’ («точка»):

struct Stack { // Cтек

float arr[100];

short topIndex;

};

…

Stack stack; // Объявляем переменную типа Stack

Stack.arr[0] = 1;

…

При доступе к определенному полю его следует рассматривать как обычную переменную, тип данных которой соответствует типу этого поля. Поля структур могут участвовать в качестве операндов любых выражений, допускающих использование операндов соответствующего типа данных.

Копирование данных из одной структурной переменной в другую осуществляется простой операцией присваивания, независимо от количества полей и размера структуры (это можно делать только в том случае, когда обе переменные одного и того же типа).

В программировании очень часто используются такие конструкции, как массивы структур. Например, сведения о студентах некоторой учебной группы можно хранить в массиве студентов:

t\_Student Gruppa\_N [30];

Был определен 30-элементный массив, каждый элемент которого предназначен для хранения данных одного студента. Получение доступа к данным некоторого студента из группы *N* осуществляется обычной индексацией переменной массива. Поскольку поля структуры могут быть любого типа данных, то они в свою очередь могут быть другой структурой или массивом других структур:

struct Stud

{

char FN[100];

short listNumber;

};

struct Group

{

int groupNumber;

short students;

Stud stud[30];

};

Но в структуре поля нельзя использовать элемент, тип которого совпадает с типом самой структуры, так как рекурсивное использование структур запрещено.

Любая структурная переменная занимает в памяти определенное положение, характеризующееся конкретным адресом. Для работы с адресами структурных переменных (как и для простых переменных) можно использовать указатели. Указатели на структурные переменные определяются точно так же, как и для обычных переменных. Разыменование указателя (обращение к данным по адресу, хранящемуся в указателе) осуществляется также обычным образом.

Через указатели можно работать с отдельными полями структур. Для доступа к полю структуры через указатель используется оператор ‘’ («стрелка»), а не «точка».

Структуры можно использовать в качестве параметров функций, как и обычные переменные. Для структур поддерживаются все три механизма передачи данных: по значению, через указатели и по ссылке.

Передачу структур в функции по значению необходимо использовать аккуратно:

void WriteStudent ( t\_Student S )

{

cout << "Фамилия: " << S.Fam << endl;

cout << "Имя: " << S.Name << endl;

cout << "Год рождения: " << S.Year << endl;

if ( S.Sex )

cout << "Пол: " << "М\n";

else

cout << "Пол: " << "Ж\n";

cout << "Средний балл: " << S.Grade << endl;

}

Вызов такой функции сопровождается дополнительным расходом памяти для создания локальной переменной *S*и дополнительными затратами времени на физическое копирование данных из аргумента в параметр *S*. Учитывая то, что объем структур может быть очень большим, эти дополнительные затраты вычислительных ресурсов могут быть чрезмерными.

Предпочтительно использование передачи структуры по указателю или ссылке:

void WriteStudent ( t\_Student \*S )

{

cout << "Фамилия: " << S -> Fam << endl;

cout << "Имя: " << S -> Name << endl;

cout << "Год рождения: " << S -> Year << endl;

if ( S -> Sex )

cout << "Пол: " << "М\n";

else

cout << "Пол: " << "Ж\n";

cout << "Средний балл: " << S -> Grade << endl;

}

Фактической передачи данных в функцию не осуществляется. Дополнительные затраты памяти для создания локальной переменной небольшие – это адрес памяти (4 байта, независимо от размера самой структуры). Вызов такой функции будет происходить быстрее, а расход памяти будет существенно меньше, чем при передаче данных по значению.

Передача по ссылке по эффективности эквивалентна передаче данных через указатель. Однако, поскольку при передаче данных по ссылке все адресные преобразования берет на себя компилятор, существенно упрощается программирование действий со структурами. При использовании ссылочных параметров структурных типов доступ к членам структуры осуществляется обычным способом – с помощью оператора «точка».

Недостатком этих способов является то, что случайные изменения значений полей структуры внутри функции отразятся на значении аргумента после окончания работы функции. Если необходимо предотвратить изменения переданных по адресу аргументов, можно при определении соответствующего параметра объявить его константой (использовать спецификатор const).

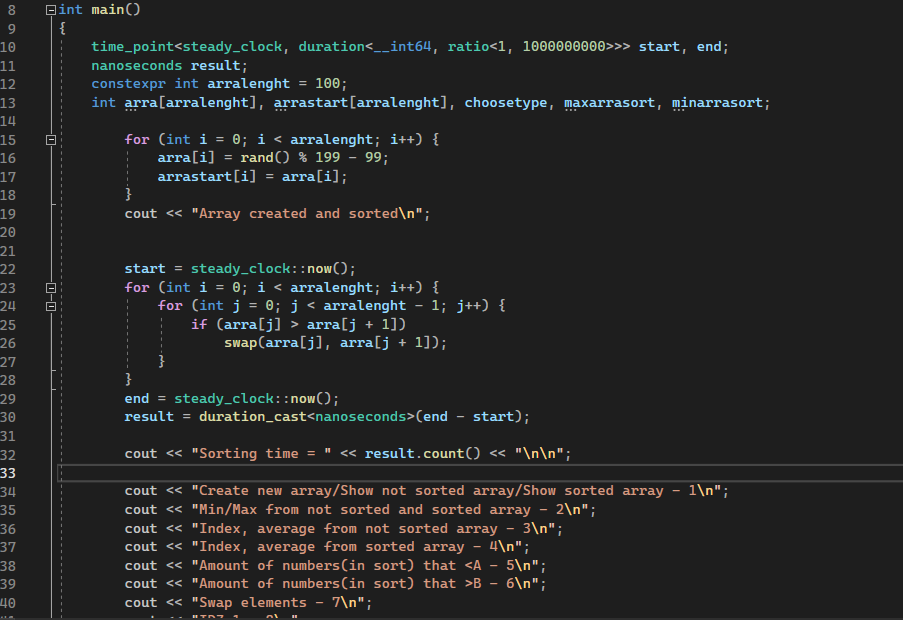
**Постановка задачи.**

* 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.
* Таблица – Варианты сортировок

|  |  |
| --- | --- |
| * Номер варианта | * Название сортировки |
| * 1 | * Bubble sort (пузырьковая сортировка) |
| * 2 | * Shaker sort (шейкер-сортировка) |
| * 3 | * Comb sort (сортировка расчёской) |
| * 4 | * Insert sort (сортировка вставками) |
| * 5 (\*) | * Quick sort (быстрая сортировка) |

**Выполнение работы.**

**Код программы представлен в приложении А**.



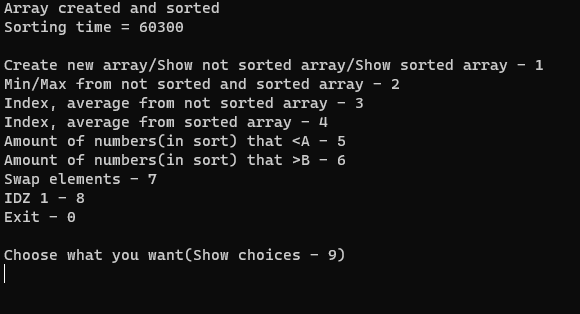
Часть кода

**Блок описания кода и использованных алгоритмов**

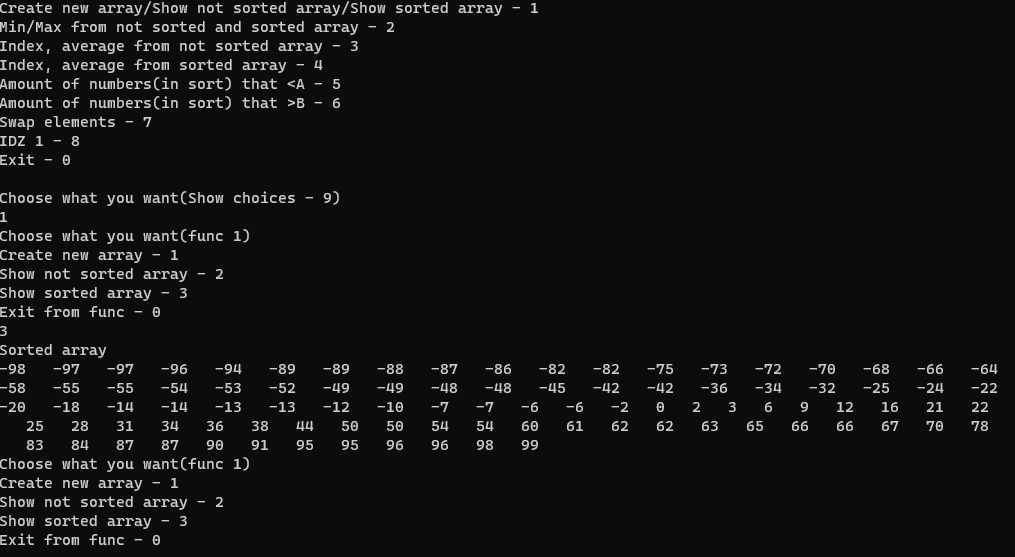
Использованы switch, do, bubble сортировка, cout, cin, if. Это очень нужные конструкции, без которых бы данный код не заработал!!!

**Блок скриншотов работы программы**

Выбор задания



**Блок таблицы с тестовыми данными**



**Выводы.**

Выполнив данную работу мы поняли, как долго выполняет сортировку компьютер, что сортировка может ускорить или замедлить работу программы, как отсортировать массив. Так же мы усовершенствовали свои навыки использования конструкций if, else, работы с массивами

Приложение А

рабочий код

#include <iostream>

#include <chrono>

#include <time.h>

#include <algorithm>

using namespace std;

using namespace chrono;

int main()

{

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

nanoseconds result;

constexpr int arralenght = 100;

int arra[arralenght], arrastart[arralenght], choosetype, maxarrasort, minarrasort;

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

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

arrastart[i] = arra[i];

}

cout << "Array created and sorted\n";

start = steady\_clock::now();

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

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

if (arra[j] > arra[j + 1])

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

}

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "Sorting time = " << result.count() << "\n\n";

cout << "Create new array/Show not sorted array/Show sorted array - 1\n";

cout << "Min/Max from not sorted and sorted array - 2\n";

cout << "Index, average from not sorted array - 3\n";

cout << "Index, average from sorted array - 4\n";

cout << "Amount of numbers(in sort) that <A - 5\n";

cout << "Amount of numbers(in sort) that >B - 6\n";

cout << "Swap elements - 7\n";

cout << "IDZ 1 - 8\n";

cout << "Exit - 0\n\n";

do {

cout << "Choose what you want(Show choices - 9)\n";

cin >> choosetype;

switch (choosetype)

{

case 1: {

int choosetypearra;

do {

cout << "Сhoose what you want(func 1)\n" << "Create new array - 1\n" << "Show not sorted array - 2\n" << "Show sorted array - 3\n" << "Exit from func - 0\n";

cin >> choosetypearra;

switch (choosetypearra)

{

case 1: {

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

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

arrastart[i] = arra[i];

}

cout << "New array created and sorted\n";

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

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

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

if (arra[j] > arra[j + 1])

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

}

}

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

auto result = duration\_cast<nanoseconds>(end - start);

cout << "Sorting time = " << result.count() << "\n";

break;

}

case 2: {

cout << "Not sorted array\n";

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

{

cout << arrastart[i] << " ";

}

cout << "\n";

break;

}

case 3: {

cout << "Sorted array\n";

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

cout << arra[i] << " ";

}

cout << "\n";

break;

}

}

} while (choosetypearra);

break;

}

case 2: {

cout << "You choosed show Min/Max from not sorted and sorted array(2)\n";

int minarrastart = arrastart[0], maxarrastart = arrastart[0];

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

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

if (arrastart[i] > maxarrastart) {

maxarrastart = arrastart[i];

}

if (arrastart[i] < minarrastart) {

minarrastart = arrastart[i];

}

}

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

auto result = duration\_cast<nanoseconds>(end - start);

cout << "Search time of not sorted array = " << result.count() << "\n";

cout << "Min from not sort = " << minarrastart << "\n";

cout << "Max from not sort = " << maxarrastart << "\n";

int maxarrasort = arra[0], minarrasort = arra[0];

auto start1 = steady\_clock::now();

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

if (arra[i] > maxarrasort) {

maxarrasort = arra[i];

}

if (arra[i] < minarrasort) {

minarrasort = arra[i];

}

}

auto end1 = steady\_clock::now();

auto result1 = duration\_cast<nanoseconds>(end1 - start1);

cout << "Search time after bubble = " << result1.count() << "\n";

cout << "Min from sort= " << minarrasort << "\n";

cout << "Max from sort= " << maxarrasort << "\n"<<"(2) ended\n\n";

break; }

case 3: {

cout << "You choosed show index, average from not sorted array(3)\n";

int minarrastartn = arrastart[0], maxarrastartn = arrastart[0], countindexmax = 0, countindexmin = 0;

float averagenotsort;

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

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

if (arrastart[i] > maxarrastartn) {

maxarrastartn = arrastart[i];

}

if (arrastart[i] < minarrastartn) {

minarrastartn = arrastart[i];

}

}

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

if (arrastart[i] == minarrastartn) {

cout << "With index " << i << " contained element with min value\n";

countindexmin += 1;

}

if (arrastart[i] == maxarrastartn) {

cout << "With index " << i << " contained element with max value\n";

countindexmax += 1;

}

}

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

auto result = duration\_cast<nanoseconds>(end - start);

cout << "Count min = " << countindexmin << "\n" << "Count max = " << countindexmax << "\n";

cout << "Average search of not sorted array time = " << result.count() << "\n";

averagenotsort = (minarrastartn + maxarrastartn) / 2.0;

cout << "Average from not sort= " << averagenotsort << "\n";

cout << "(3) ended\n\n";

break;

}

case 4: {

cout << "You choosed show index, average from sorted array(4)\n";

int maxarrasortn = arra[0], minarrasortn = arra[0], countindexmax = 0, countindexmin = 0;

float averagesort;

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

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

if (arra[i] > maxarrasortn) {

maxarrasortn = arra[i];

}

if (arra[i] < minarrasortn) {

minarrasortn = arra[i];

}

}

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

if (arra[i] == minarrasortn) {

cout << "With index " << i << " contained element with min value\n";

countindexmin += 1;

}

if (arra[i] == maxarrasortn) {

cout << "With index " << i << " contained element with max value\n";

countindexmax += 1;

}

}

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

auto result = duration\_cast<nanoseconds>(end - start);

cout << "Count min = " << countindexmin << "\n" << "Count max = " << countindexmax << "\n";

cout << "Average search of sorted array time = " << result.count() << "\n";

averagesort = (minarrasortn + maxarrasortn) / 2.0;

cout << "average from sort= " << averagesort << "\n";

cout << "(4) ended\n\n";

break;

}

case 5: {

cout << "You choosed show mount of numbers(in sort) that <A(5)\n";

int numbera, countindexa=0;

cout << "Initialize the number A\n";

cin >> numbera;

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

if (arra[i] < numbera) {

countindexa += 1;

}

}

cout << "Amount of numbers that <A = " << countindexa << "\n";

cout << "(5) ended\n\n";

break;

}

case 6: {

cout << "You choosed show amount of numbers(in sort) that >B(6)\n";

int numbera, countindexa = 0;

cout << "Initialize the number B\n";

cin >> numbera;

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

if (arra[i] > numbera) {

countindexa += 1;

}

}

cout << "Amount of numbers that >B = " << countindexa << "\n";

cout << "(6) ended\n\n";

break;

}

case 7: {

cout<<"You choosed swap elements(7)\n";

int choosearra, indexfirst, indexsecond;

cout << "Choose in which array we will swap( 1 - sort, 2 - not sort, 0 - exit)\n";

cin >> choosearra;

if (choosearra == 0)

break;

cout << "Enter first index of element to swap\n";

cin >> indexfirst;

cout << "Enter second index of element to swap\n";

cin >> indexsecond;

switch (choosearra) {

case 1: {

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

swap(arra[indexfirst], arra[indexsecond]);

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

auto result = duration\_cast<nanoseconds>(end - start);

cout << "Time of swap = " << result.count() << "\n";

break;

}

case 2: {

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

swap(arrastart[indexfirst], arrastart[indexsecond]);

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

auto result = duration\_cast<nanoseconds>(end - start);

cout << "Time of swap = " << result.count() << "\n";

break;

}

} //switch end

cout << "(7) ended\n";

break;

}

case 8: {

cout << "For not sorted array\n";

int newarra[arralenght],count[10];

for (int i = 0; i < 10; i++)

count[i] = 0;

newarra[arralenght - 1] = arrastart[0] + arrastart[arralenght - 1];

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

newarra[i] = arrastart[i] + arrastart[i + 1];

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

swap(newarra[i], newarra[rand() % 100]);

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

for (int j = 1; j <= 9; j++) {

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

count[j] += 1;

}

}

}

for (int j = 1; j <= 9; j++)

cout << "Amount of elements who divided entirely into " << j << " = " << count[j] << "\n";

break;

}

case 9: {

cout << "Create new array/Show not sorted array/Show sorted array - 1\n";

cout << "Min/Max from not sorted and sorted array - 2\n";

cout << "Index, average from not sorted array - 3\n";

cout << "Index, average from sorted array - 4\n";

cout << "Amount of numbers(in sort) that <A - 5\n";

cout << "Amount of numbers(in sort) that >B - 6\n";

cout << "Swap elements - 7\n";

cout << "Idz - 8 \n";

cout << "Exit - 0\n";

break;

}

}//switch end

} while (choosetype);

return 0;

}