Белорусский государственный университет информатики и радиоэлектроники

Факультет компьютерных технологий

ОТЧЕТ

по лабораторной работе №2

на тему: **Алгоритмы посика и сортировки в массивах**

|  |  |
| --- | --- |
| Выполнил  студент  группы №980161  Алейчик И.Д. | Проверил  доцент  Казакевич Ю.Д. |

Минск 2020

**Лаборатоорная работа №2**

**Вариант 1**

**Тема:**

Алгоритмы поиска и сортировки в массивах

**Цель работы:**

Изучить способы сортировки и поиска в массивах структур и файлах.

**Краткие теоретические сведения:**

При обработке баз данных часто применяются массивы структур.Обычно база данных накапливается и хранится на диске в файле. К ней часто приходит-ся обращаться, обновлять, перегруппировывать. Работа с базой может быть ор-ганизована двумя способами.

1. Внесение изменений и поиск осуществляются прямо на диске, с ис-пользованиемспецифическойтехникиработы со структурами в файлах. При этом временные затраты на обработку данных (поиск, сортировку) значительно возрастают, но нет ограничений на использование оперативной памяти.

2. Считывание всей базы (или необходимой ее части) в массив структур. При этом обработка производится в оперативной памяти, что значительно уве-личивает скорость, однако требует больших затрат памяти.

Наиболее частыми операциями при работе с базами данных являются «поиск» и «сортировка». При этом алгоритмы решения этих задач существенно зависят от того, организованы структуры в массивы или размещены на диске.

Обычно элемент данных (структура) содержит некое ключевое поле (ключ),по которому его можно найти. Ключом может служить любое поле структуры, например, фамилия, номер телефона или адрес. Основное требова-ние к ключу в задачах поиска состоит в том,чтобы операция проверки на ра-венство была корректной, поэтому при поиске данных по ключу, имеющему вещественное значение, следует указывать не его конкретное значение, аин-тервал, в который это значение попадает.

**Задание:**

Написать программу обработки файла данных, состоящих из структур, в которой реализованы следующие функции: стандартная обработка файла (со-здание, просмотр, добавление); линейный поиск в файле; сортировка массива (файла) методами прямого выбора и QuickSort; двоичный поиск в отсортиро-ванном массиве.

1. В магазине формируется список лиц, записавшихся на покупку товара. Вид списка: номер, ФИО, домашний адрес,датаучета. Удалить из списка все повторяющиесязаписи, проверяя ФИО и адрес. Ключ: дата постановки на учет.

**Ход выполнения работы:**

#include <iostream>

#include <Windows.h>

#include <string>

#include <regex>

#include <fstream>

#include <conio.h>

#include <iomanip>

#include <time.h>

using namespace std;

struct Customer

{

int id;

string \_surname;

string \_name;

string \_patronymic;

string \_home\_adress;

string \_accounting\_date;

};

string InputSubscribeOwner = { "InputSubscribeOwner.txt" };

void qsortRecursive(Customer\* mas, int size) {

//Определяем начало и в конец массива

int i = 0;

int j = size -1;

//Центральный элемент массива

int mid = size / 2;

//Делим массив

do {

//в левой части массива оставляем элементы на месте, которые меньше центрального

while (mas[i].id < mas[mid].id) {

i++;

}

//в правой части пропускаем элементы, которые больше центрального

while (mas[j].id > mas[mid].id) {

j--;

}

//меняем элементы местами

if (i <= j) {

Customer tmp = mas[i];

mas[i] = mas[j];

mas[j] = tmp;

i++;

j--;

}

} while (i <= j);

//Рекурсивные вызовы, на случай если что то осталось

if (j > 0) {

//"Левый кусок"

qsortRecursive(mas, j + 1);

}

if (i < size) {

//"Првый кусок"

qsortRecursive(&mas[i], size - i);

}

}

//линейный посик

int linearSearch(Customer\* mas,int size,string key) {

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

{

if (mas[i].\_accounting\_date == key)

{

return i;

}

}

}

//сортировка прямым выбором

void sortDirectSelection(Customer \* mas, int size) {

int min;

Customer temp; // для поиска минимального элемента и для обмена

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

{

min = i; // запоминаем индекс текущего элемента

// ищем минимальный элемент чтобы поместить на место i-ого

for (int j = i + 1; j < size; j++) // для остальных элементов после i-ого

{

if (mas[j].id < mas[min].id) // если элемент меньше минимального,

min = j; // запоминаем его индекс в min

}

temp = mas[i]; // меняем местами i-ый и минимальный элементы

mas[i] = mas[min];

mas[min] = temp;

}

}

//подсчет кол-ва элементов из файла

int countFromFile(string nameFile) {

int k = 0;

ifstream fin(nameFile);

if (fin.is\_open()) {

fin >> k;

}

else {

cout << "Ошибка открытия файла.\n";

}

fin.close();

return k;

}

//добовление в файл

void AddToFile(string nameFile, Customer\* C, int i, int id)

{

ofstream fout(nameFile, std::ios::app);

if (fout.is\_open())

{

fout << C[i].id << " "<< C[i].\_surname << " " << C[i].\_name << " " << C[i].\_patronymic << "\n" << C[i].\_home\_adress << "\n" << C[i].\_accounting\_date << "\n";

}

else { cout << "Ошибка открытия файла\n"; }

}

//запись в файл данных абонента

void writeInFileInput(int n, string nameF, Customer\* C) {

ofstream fout(nameF, std::ios::out);

if (fout.is\_open())

{

fout << n << endl;

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

{

AddToFile(nameF, C,i,n);

}

}

else { cout << "Ошибка открытия файла \"O/F.txt\".\n"; }

fout.close();

}

//чтение из файла данных абонента

Customer\* readFromFile(Customer\* S, string nameFile) {

int k, Id;

string Surname, Name, Patronymic, Adress, Date;

ifstream fin(nameFile); //читаем файл

//заполняем временные переменные данными из файла

if (fin.is\_open()) {

fin >> k;

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

fin >> Id >> Surname >> Name >> Patronymic >> Adress >> Date;

S[i].id = Id;

S[i].\_surname = Surname;

S[i].\_name = Name;

S[i].\_patronymic = Patronymic;

S[i].\_home\_adress = Adress;

S[i].\_accounting\_date = Date;

}

}

return S;

}

//функция бинарного поиска

int binarySearch(Customer\* S, string key, int size)

{

int left = 0, right = size, mid; // устанавливаем границы

readFromFile(S, InputSubscribeOwner); // считываем данные файйла в массив

while (left <= right)

{

mid = left + (right - left) / 2; // определяем средний эдемент/центр

if (key < S[mid].\_accounting\_date) right = mid - 1; // если искомый элемент меньше, сдвигаем правую границу влево от срденего жлемента на 1

else if (key > S[mid].\_accounting\_date) left = mid + 1; // если искомый элемент больше, сдвигаем левую границу вправо от срденего жлемента на 1

else return mid; // иначе возвращаем найденный элемент

}

system("pause");

}

//вывод конкретного элемента списка (перегруженная функция)

void viewData(int countSubcsribe, string nameInputFile, int index) {

countSubcsribe = countFromFile(nameInputFile);

Customer\* S = new Customer[countSubcsribe];

readFromFile(S, nameInputFile);

cout << setw(15) << "ID" << setw(30) << "ФИО" << setw(15) << "Адрес" << setw(15) << "Дата учета" << "\n";

if (S[index].\_surname != "") {

cout << setw(15) << S[index].id << setw(30) << S[index].\_surname + " " + S[index].\_name + " " + S[index].\_patronymic << setw(15) << S[index].\_home\_adress << setw(15) << S[index].\_accounting\_date << "\n";

}

\_getch();

}

//вывод списка с удалением повторений

void viewData(int countSubcsribe, string nameInputFile) {

countSubcsribe = countFromFile(nameInputFile);

Customer\* S = new Customer[countSubcsribe];

readFromFile(S, nameInputFile);

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

{

for (int j = i + 1; j < countSubcsribe; j++)

{

if ((S[i].\_surname == S[j].\_surname) && (S[i].\_name == S[j].\_name) && (S[i].\_patronymic == S[j].\_patronymic) && S[i].\_home\_adress == S[j].\_home\_adress) // если найден одинаковый элемент

{

for (int c\_shift = j; c\_shift < countSubcsribe - 1; c\_shift++)

{

// сдвиг всех остальных элементов массива на -1, начиная со следующего элемента, после найденного дубля

S[c\_shift] = S[c\_shift + 1];

}

countSubcsribe -= 1; // уменьшить размер массива на 1

if (S[i].\_home\_adress == S[j].\_home\_adress) // если следующий элемент - дубль

{

j--; // переход на предыдущий элемент

}

}

}

}

//вывод списка

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

{

cout << setw(15) << "ID" << setw(30) << "ФИО" << setw(15) << "Адрес" << setw(15) << "Дата учета" << "\n";

if (S[i].\_surname != "") {

cout << setw(15) << S[i].id << setw(30) << S[i].\_surname +" "+ S[i].\_name +" "+ S[i].\_patronymic << setw(15) << S[i].\_home\_adress << setw(15) << S[i].\_accounting\_date << "\n";

}

}

writeInFileInput(countSubcsribe, nameInputFile, S);

\_getch();

}

void addData(int countSubcsribe, string nameInputFile) {

string Surname, Name, Patronymic,Adress,Date;

// регулярные выражения для проверки валидности вводимых данных

regex date(R"((0[1-9]|[12][0-9]|3[01])[- /.](0[1-9]|1[012])[- /.](19|20)\d\d)");

regex phone("[+](375)[0-9]{2}[0-9]{7}");

regex year("[0-9]{4}");

regex text("^[а-яА-ЯёЁa-zA-Z]+$");

int NewCountSubscribeOwner = countSubcsribe + 1;

Customer\* S = new Customer[NewCountSubscribeOwner];

readFromFile(S, nameInputFile);

bool isValueName = false;

cin.ignore(32767, '\n');

do

{

isValueName = false;

cout << "Введите фамилию: ";

getline(cin, Surname);

if (Surname == "") {

cout << "Строка неможет быть пустой\n";

isValueName = true;

}

else {

if (regex\_match(Surname, text)) { isValueName = false; }

else {

cout << "ФИО может содерать только знаки кирилицы и/или знаки латинского алфавита \n";

isValueName = true;

}

}

} while (isValueName != false);

do {

isValueName = false;

cout << "Введите имя: ";

getline(cin, Name);

if (Name == "") {

cout << "Строка неможет быть пустой\n";

isValueName = true;

}

else {

if (regex\_match(Name, text)) { isValueName = false; }

else {

cout << "ФИО может содерать только знаки кирилицы и/или знаки латинского алфавита \n";

isValueName = true;

}

}

} while (isValueName != false);

do {

isValueName = false;

cout << "Введите отчество: ";

getline(cin, Patronymic);

if (Patronymic == "") {

cout << "Строка неможет быть пустой\n";

isValueName = true;

}

else {

if (regex\_match(Patronymic, text)) { isValueName = false; }

else {

cout << "ФИО может содерать только знаки кирилицы и/или знаки латинского алфавита \n";

isValueName = true;

}

}

} while (isValueName != false);

do {

isValueName = false;

cout << "Введите домашний адрес: ";

getline(cin, Adress);

if (Adress == "") {

cout << "Строка неможет быть пустой\n";

isValueName = true;

}

else {

if (regex\_match(Adress, text)) { isValueName = false; }

else {

cout << "Адрес может содерать только знаки кирилицы и/или знаки латинского алфавита \n";

isValueName = true;

}

}

} while (isValueName != false);

do {

isValueName = false;

cout << "Введите дату поставки на учет ";

getline(cin, Date);

if (Date == "") {

cout << "Строка неможет быть пустой\n";

isValueName = true;

}

else {

if (regex\_match(Date, date)) { isValueName = false; }

else {

cout << "Адрес может содерать только цифры и спец символы типа точки (.) \n";

isValueName = true;

}

}

} while (isValueName != false);

//заполняем новый элемент массива

S[NewCountSubscribeOwner - 1].id = NewCountSubscribeOwner;

S[NewCountSubscribeOwner - 1].\_surname = Surname;

S[NewCountSubscribeOwner - 1].\_name = Name;

S[NewCountSubscribeOwner - 1].\_patronymic = Patronymic;

S[NewCountSubscribeOwner - 1].\_home\_adress = Adress;

S[NewCountSubscribeOwner - 1].\_accounting\_date = Date;

//выводим

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

cout << "Абонент №" << i + 1 << " :\n";

if (S[i].\_surname != "") {

cout << "ФИО :" << S[i].\_surname << " " << S[i].\_name << " " << S[i].\_patronymic << endl;

}

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

}

//записываем в файл

writeInFileInput(NewCountSubscribeOwner, nameInputFile, S);

cout << "успешно добавлен\n";

\_getch();

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL,"");

string inp\_str;

int s;

cout << "МЕНЮ\n"

<< " (1) Добовление данных\n"

<< " (2) Вывод информации\n"

<< " (3) Quick Sort\n"

<< " (4) Direct Selection Sort\n"

<< " (5) Linear Search\n"

<< " (6) Binary Search\n";

cout << "->";

cin >> s;

while (s != 0) {

//добавить данные о покупателе

if (s == 1) {

addData(countFromFile(InputSubscribeOwner), InputSubscribeOwner);

return 0;

}

//вывести данные покупателей

if (s == 2) {

viewData(countFromFile(InputSubscribeOwner), InputSubscribeOwner);

return 0;

}

//быстрая сортировка

if (s == 3) {

int n = countFromFile(InputSubscribeOwner);

Customer\* S = new Customer[n];

readFromFile(S, InputSubscribeOwner);

qsortRecursive(S, n);

writeInFileInput(n, InputSubscribeOwner, S);

viewData(countFromFile(InputSubscribeOwner), InputSubscribeOwner);

system("pause");

return 0;

}

//сортировка прямым выбором

if (s == 4)

{

Customer\* S = new Customer[countFromFile(InputSubscribeOwner)];

readFromFile(S, InputSubscribeOwner);

sortDirectSelection(S, countFromFile(InputSubscribeOwner));

writeInFileInput(countFromFile(InputSubscribeOwner), InputSubscribeOwner, S);

viewData(countFromFile(InputSubscribeOwner), InputSubscribeOwner);

system("pause");

}

//линейный поиск

if (s == 5) {

cin >> inp\_str;

Customer\* S = new Customer[countFromFile(InputSubscribeOwner)];

readFromFile(S, InputSubscribeOwner);

viewData(0, InputSubscribeOwner, linearSearch(S, countFromFile(InputSubscribeOwner), inp\_str));

return 0;

}

//бинарный поиск

if (s == 6) {

cin >> inp\_str;

Customer\* S = new Customer[countFromFile(InputSubscribeOwner)];

readFromFile(S, InputSubscribeOwner);

viewData(0, InputSubscribeOwner, binarySearch(S, inp\_str, countFromFile(InputSubscribeOwner)));

return 0;

}

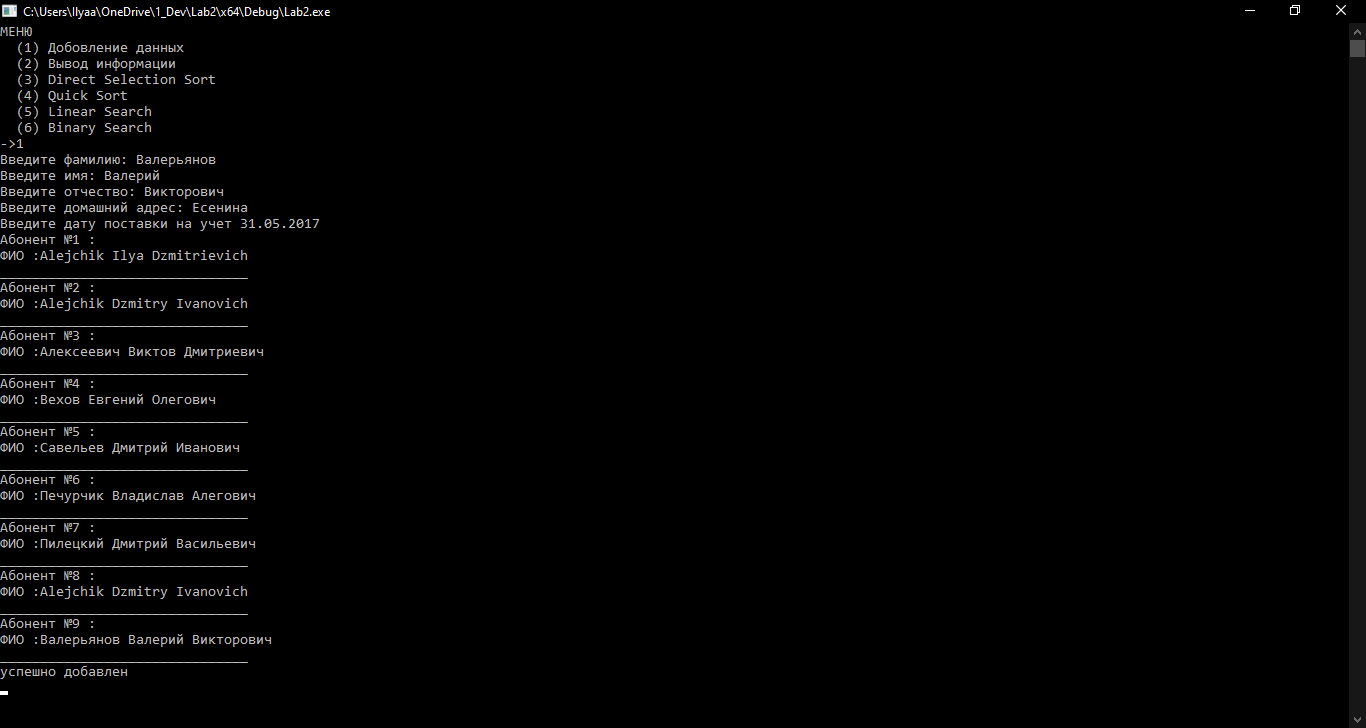
}

exit(0);

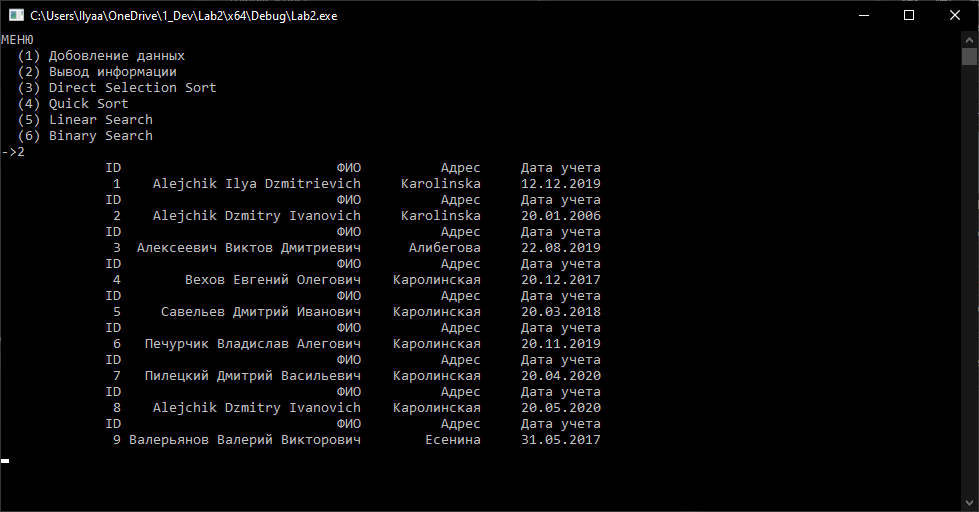
}

**Результат работы программы:**

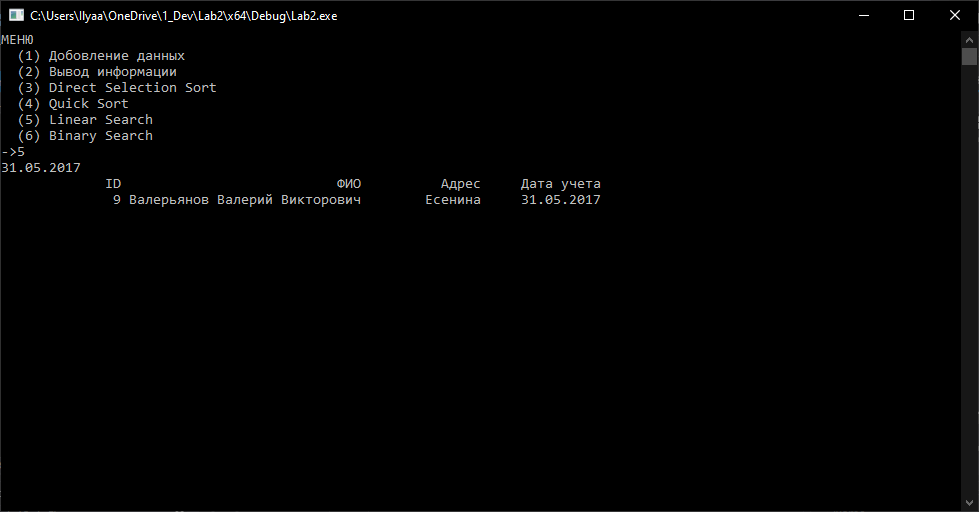
Добавление



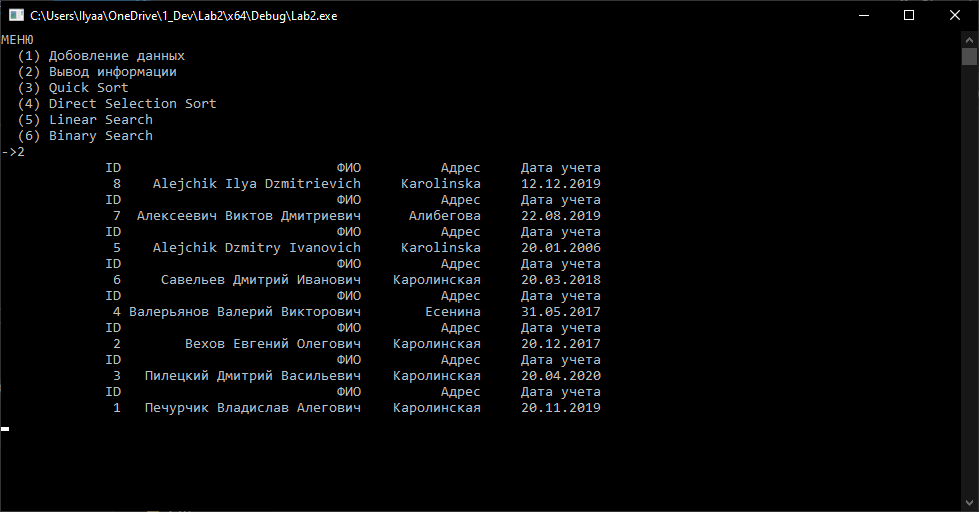
Просмотр

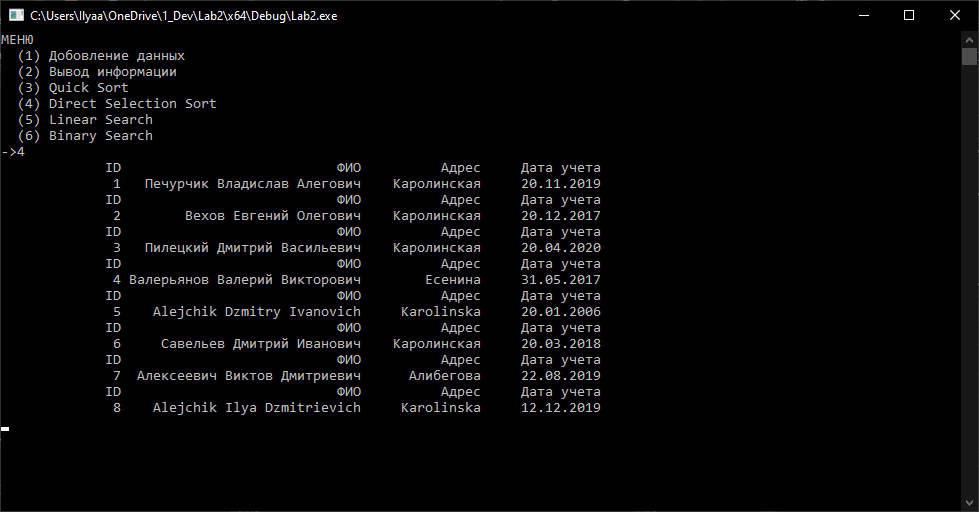


Линейный поиск

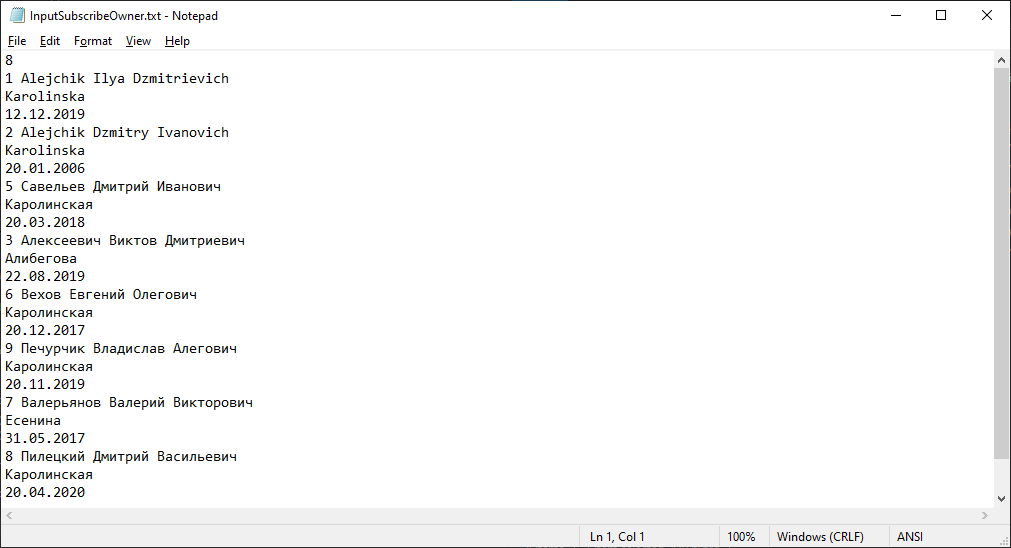


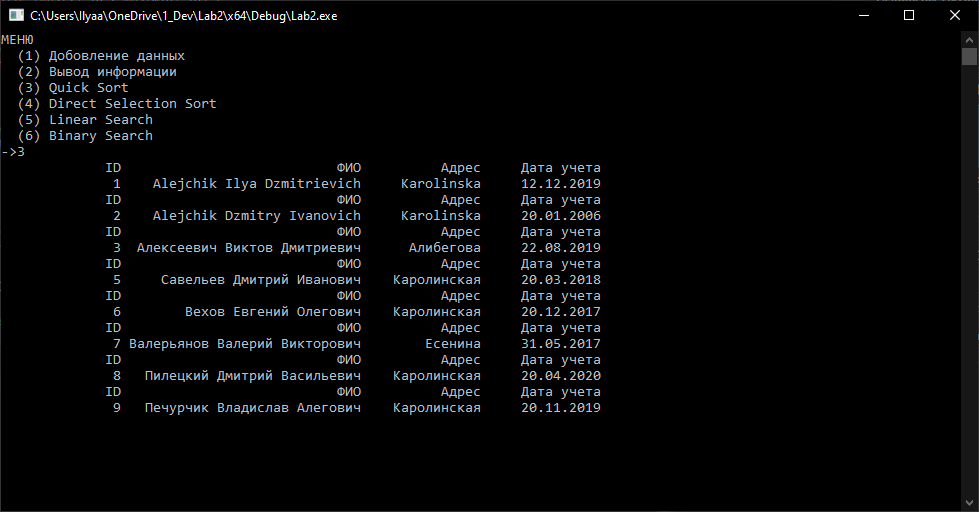
Сортировка методом прямого выбора



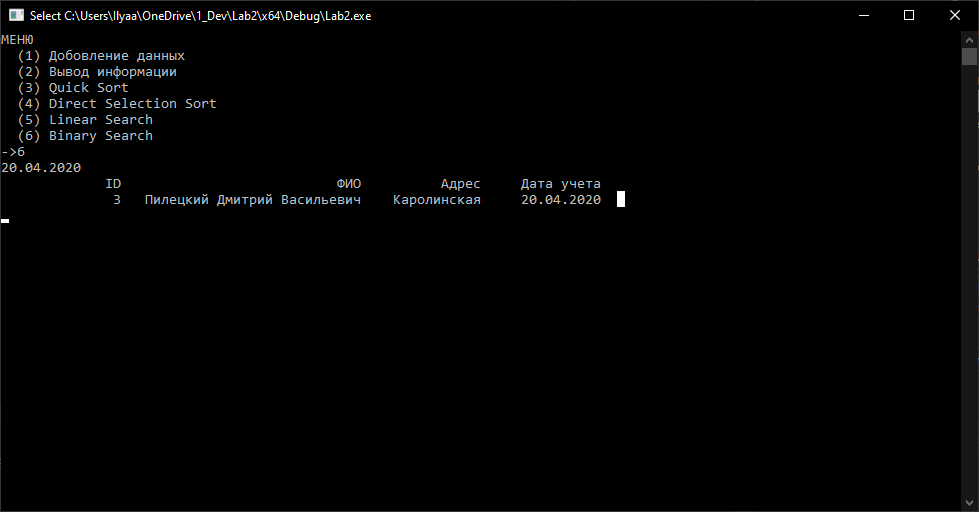


Быстрая сортировка





Бинарный поиск в отсортированном массиве



**Вывод:**

Были изучены способы сортировки и посика в массивах структур и файлах.