МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ЭТПТ

КУРСОВАЯ РАБОТА

по дисциплине «Информационные технологии»

Тема: Система управление базой данных

Студент гр. 3403 Матюхина А.А.

Преподаватель Чмиленко Ф.В

Санкт-Петербург
2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Получение навыков постановки задачи, алгоритмизации, модульного принципа разработки и отладки приложений на примере создания программы для работы с базой данных в виде типизированного файла.

Создать программу, которая работает с базой данных в виде типизированного файла. Код программы должен поддерживать модульный принцип разработки (состоять не менее чем из трех модулей). При реализации программы необходимо использовать функции, массивы, структуры, указатели, выполнять форматное преобразование данных, чтение и запись в файлы.

1. ОПИСАНИЕ МОДУЛЕЙ ПРОГРАММЫ

1.1. Файл «main.cpp»

Основная функция программы, обрабатывающая запрос пользователя и распределяющая его передающая его вспомогательным функциям.

1.2. Файл «trains.cpp» и «trains.h»

Содержат в себе тела и заголовки вспомогательных функций, производящих основные манипуляции с массивом, файлами и потоком вывода консоли.

2. ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

2.1. Функция «void addRecord(Train*& trains, int& numTrains, int& MAX_LIST_LENGTH)»

Принимает указатель на массив структуры Train, содержащий информацию о элементах таблицы, количество записей в массиве, а также текущий объём памяти, выделенный под массив. Не возвращает значение. Добавляет элемент в массив структуры Train.

2.2. Функция «void readDataFromFile(Train*& trains, int& numTrains, int& MAX_LIST_LENGTH)»

Принимает массив структуры Train, количество записей в нём и изначально заданное количество допустимых записей, указанных пользователем. Не возвращает значение. Читает данные из указанного пользователем файла.

2.3. Функция «void displayData(const Train trains[], int numTrains)»

Принимает массив структуры Train, а также количество записей в нём. Не возвращает значение. Выводит в консоль данные массива структуры Train в виде таблицы.

2.4. Функция «void editRecord(Train trains[], int numTrains)»

Принимает массив структуры Train, а также количество записей в нём. Не возвращает значение. Редактирует выбранный пользователем элемент массива структуры Train.

2.5. Функция «void saveData(const Train trains[], int& numTrains)»

Принимает массив структуры Train, а также количество записей в нём. Не возвращает значение. Сохраняет данные массива структуры Train в указанный пользователем файл.

2.6. Функция «void deleteRecord(Train trains[], int& numTrains)»

Принимает массив структуры Train, а также количество записей в нём.Не возвращает значение. Удаляет указанные пользователем данные из массива структуры Train.

2.7. Функция «void sortRecords(Train trains[], int numTrains)»

Принимает массив структуры Train, а также количество записей в нём. Не возвращает значение. Сортирует массив структуры Train по выбранному пользователем полю, на основании выбраного пользователем режима сортировки.

2.8. Функция «double findAverageTicketPrice(const Train trains[], int numTrains)»

Принимает массив структуры Train, а также количество записей в нём. Возвращает значение с плавающей запятой. Возвращает значение средней цены за билет.

2.9. Функция «double findMaxTicketPrice(const Train trains[], int numTrains)»

Принимает массив структуры Train, а также количество записей в нём. Возвращает значение с плавающей запятой. Возвращает значение наивысшей цены за билет.

2.10. Функция «void searchRecord(const Train trains[], int numTrains)»

Принимает массив структуры Train, а также количество записей в нём. Не возвращает значение. Осуществляет поиск по введенному пользователем ключевому слову.

ЗАКЛЮЧЕНИЕ

В ходе выполнения задачи получила навыки постановки задачи, алгоритмизации, модульного принципа разработки и отладки приложений на примере создания программы для работы с базой данных в виде типизированного файла.

А также создала программу, которая работает с базой данных в виде типизированного файла. Код программы поддерживает модульный принцип разработки. При реализации программы использовала функции, массивы, структуры, указатели, выполнила форматное преобразование данных, чтение и запись в файлы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Чмиленко Ф. В., Бондарь А. С., Хоршев А. А. ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ С++. СПб.: Издательство СПбГЭТУ «ЛЭТИ», 2021. 39 с.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

main.cpp

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include "train.h"
using namespace std;
int main() {
	setlocale(LC_ALL, "");
	SetConsoleCP(1251);
	bool dataModified = false;
	int MAX_LIST_LENGTH = 10;
	int numTrains = 0;
	Train* trains = new Train[MAX_LIST_LENGTH];
	Train* trains_copy = trains;
	int choice;  //номер операции
	bool continueRunning = true;
	do {
		displayData(trains, numTrains);
		cout << "Меню:\n"
			"1. Чтение данных из файла\n"
			"2. Вывод данных на экран\n"
			"3. Редактирование записи\n"
			"4. Сохранение данных в файл\n"
			"5. Удаление записи\n"
			"6. Сортировка записей\n"
			"7. Поиск записи по названию и компании\n"
			"8. Подсчет средней цены билета\n"
			"9. Поиск максимальной цены билета\n"
			"10. Добавление записи\n"
			"0. Выход\n"
			"Введите номер операции: ";
		cin >> choice;
		system("cls");
		switch (choice) {
		case 1:
			readDataFromFile(trains, numTrains, MAX_LIST_LENGTH);
			break;
		case 2:
			displayData(trains, numTrains);
			break;
		case 3:
			editRecord(trains, numTrains);
			dataModified = true; //флаг изменения данных
			break;
		case 4:
			saveData(trains, numTrains);
```

```
                dataModified = false;
                break;
            case 5:
                deleteRecord(trains, numTrains);
                dataModified = true; //флаг изменения данных
                break;
            case 6:
                sortRecords(trains, numTrains);
                break;
            case 7:
                searchRecord(trains, numTrains);
                break;
            case 8: {
                double averagePrice = findAverageTicketPrice(trains, numTrains);
                cout << "Средняя цена билета: " << averagePrice << endl;
                break;
            }
            case 9: {
                findTrainsWithMaxTicketPrice(trains, numTrains);
                break;
            }
            case 10:
                addRecord(trains, numTrains, MAX_LIST_LENGTH);      //предача адреса
первого элемента массива и переменной по ссылке
                dataModified = true;  //флаг изменения данных
                break;
            case 0:
                int saveChoice;
                if (dataModified) {
                    cout << "Предупреждение: данные были изменены, но не
сохранены.\n";
                    cout << "Хотите сохранить данные перед выходом? (1/2): ";
                    cin >> saveChoice;
                    if (saveChoice==1) {
                        saveData(trains, numTrains);
                    }
                }
                    dataModified = false;
                    cout << "Выход из программы.\n";
                    continueRunning = false;
                    break;
            default:
                cout << "Ошибка: неправильный номер операции." << endl;
                break;

                }

            } while (continueRunning);

            return 0;
        }
```

Train.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <iomanip>
#include <Windows.h>
using namespace std;
const int MAX_STRING_LENGTH = 100;
struct Train {
	char company[MAX_STRING_LENGTH];
	char destination[MAX_STRING_LENGTH];
	double ticket_price;
	int free_spaces;
	int code;
};
void addRecord(Train*& trains, int& numTrains, int& MAX_LIST_LENGTH);
void readDataFromFile(Train*& trains, int& numTrains, int& MAX_LIST_LENGTH);
void displayData(const Train trains[], int numTrains);
void editRecord(Train trains[], int numTrains);
void saveData(const Train trains[], int numTrains);
void deleteRecord(Train trains[], int& numTrains);
void sortRecords(Train trains[], int numTrains);
void searchRecord(const Train trains[], int numTrains);
double findAverageTicketPrice(const Train trains[], int numTrains);
void findTrainsWithMaxTicketPrice(const Train trains[], int numTrains);
```

Train.cpp

```
#include"train.h"
void addRecord(Train*& trains, int& numTrains, int& MAX_LIST_LENGTH) {
	if (numTrains >= MAX_LIST_LENGTH) {
		MAX_LIST_LENGTH *=2 ;
		Train* newTrains = new Train[MAX_LIST_LENGTH];
		memcpy(newTrains, trains, numTrains * sizeof(Train));
		delete[] trains;
		trains = newTrains;
	}
	cout << "Введите информацию о поезде:" << endl;
	Train newTrain;
	cout << "Компания: ";
	cin >> newTrain.company;
	cout << "Пункт назначения: ";
	cin >> newTrain.destination;
	cout << "Цена билета: ";
	cin >> newTrain.ticket_price;
	cout << "Свободных мест: ";
	cin >> newTrain.free_spaces;
	cout << "Id: ";
	cin >> newTrain.code;
	cout << "Хотите сохранить новую запись? (1/2) ";
	int c;
	cin >> c;
	cin.ignore();
	if(c==1){
		trains[numTrains] = newTrain;
		numTrains++;
	}

	cout << "Запись успешно добавлена." << endl;
}
void readDataFromFile(Train*& trains, int& numTrains, int& MAX_LIST_LENGTH) {
	char filename[100];
```

```
cout << "Введите имя файла для чтения данных: ";
cin >> filename;
cin.ignore();

ifstream file(filename);
if (!file) {
	cout << "Ошибка: не удалось открыть файл для чтения.\n";
	return;
}
numTrains = 0;
while (file.peek() != EOF) {
	if (numTrains >= MAX_LIST_LENGTH) {
		MAX_LIST_LENGTH *=2;
		Train* newTrains = new Train[MAX_LIST_LENGTH];
		memcpy(newTrains, trains, numTrains * sizeof(Train));
		delete[] trains;
		trains = newTrains;
	}

	Train newTrain;
	file.getline(newTrain.company, MAX_STRING_LENGTH, ';');

	file.getline(newTrain.destination, MAX_STRING_LENGTH, ';');

	file >> newTrain.ticket_price;
	file.ignore();
	file >> newTrain.free_spaces;
	file.ignore();
	file >> newTrain.code;
	file.ignore(); // Пропустить символ новой строки

	trains[numTrains] = newTrain;
	numTrains++;
}
file.close();
```

```
    cout << "Файл успешно прочитан. База данных содержит " << numTrains << " записей.\n";
}
void displayData(const Train trains[], int numTrains) { //чисто оформление
        printf("Основная таблица:\n");
        printf("|==========================================================
=======================================|\n");
        printf("| Id |       Компания       |                Пункт назначения                | Цена билета |
Свободных мест|\n");
        printf("|==========================================================
=======================================|\n");

        for (int i = 0; i < numTrains; ++i) {
                printf("| %-3d| %-18.18s | %-42.42s | %-11.2f | %-6d           | \n", trains[i].code,
trains[i].company, trains[i].destination, trains[i].ticket_price, trains[i].free_spaces);
        }

        printf("|==========================================================
=======================================|\n");
}
void editRecord(Train trains[], int numTrains) { //редактирование записей
        if (numTrains == 0) {
                cout << "Нет доступных записей." << endl;
                return;
        }
        int recordNumber;
        cout << "Введите номер записи для редактирования: ";
        cin >> recordNumber;
        if (recordNumber < 1 || recordNumber > numTrains) {
                cout << "Ошибка: неправильный номер записи." << endl;
                return;
        }
        cout << "Выберите что редактировать:\n"
                << "1.Цену \n"
                << "2.Количество свободных мест \n"
                << "3.Компания \n"
```

```
            << "4.Направление \n ";
        int choice;
        cin >> choice;
        cin.ignore();
        switch (choice) {
        case 1:
            cout << "Цена: ";
            cin >> trains[recordNumber - 1].ticket_price;
            break;
        case 2:
            cout << "Свободных мест: ";
            cin >> trains[recordNumber - 1].free_spaces;
            break;
        case 3:
            cout << "Компания: ";
            cin.getline(trains[recordNumber - 1].company, sizeof(trains[recordNumber - 1].company)); //везде минус один потому что типа мммммммммм индексация начинается с 1, поэтому вычитание 1 нужно для доступа к верному элементу массива, ибо массив с нуля начинается
            break;
        case 4:
            cout << "Пункт назначения: ";
            cin.getline(trains[recordNumber - 1].destination, sizeof(trains[recordNumber - 1].destination));
            break;
        default:
            cout << "Ошибка: неправильная опция." << endl;
            return;
            break;
        }
        cout << "Запись успешно отредактирована." << endl;
}
void saveData(const Train trains[], int numTrains) { //данные в файл
        char filename[MAX_STRING_LENGTH];
        printf("Введите имя файла для сохранения данных: ");
```

```
	scanf_s("%s", filename, 100);


	FILE* file; //указатель на структуру file
	fopen_s(&file, filename, "w");//w открытие файла для записи
	if (!file) {
		printf("Ошибка: не удалось открыть файл для сохранения.\n");
		return;
	}
	for (int i = 0; i < numTrains; i++) {
		fprintf(file, "%s;%s;%.2f;%d;%d;\n", trains[i].company, trains[i].destination,
trains[i].ticket_price, trains[i].free_spaces, trains[i].code);
	}
	fclose(file);
	printf("Данные успешно сохранены.\n");
}
void deleteRecord(Train trains[], int& numTrains) { //удаляем ))))))))))
	if (numTrains == 0) {
		cout << "Нет доступных записей." << endl;
		return;
	}
	int recordNumber;
	cout << "Введите номер записи для удаления: ";
	cin >> recordNumber;
	if (recordNumber < 1 || recordNumber > numTrains) {
		cout << "Ошибка: неправильный номер записи." << endl;
		return;
	}
	for (int i = recordNumber - 1; i < numTrains - 1; i++) {
		trains[i] = trains[i + 1];
	}
	numTrains--;
	cout << "Запись успешно удалена." << endl;
}
void sortRecords(Train trains[], int numTrains) { //сортировочка
```

```
if (numTrains == 0) {
	cout << "Нет доступных записей." << endl;
	return;
}
cout << "Выберите тип сортировки:\n"
	<< "1. По цене \n"
	<< "2. По количеству свободных мест \n"
	<< "3. По компании \n"
	<< "4. По направлению \n"
	<< "5. По номеру \n";;
int choice;
cin >> choice;
switch (choice) {
case 1: {
	cout << "Выберите тип сортировки:\n"
		<< "1. По возрастанию \n"
		<< "2. По убыванию \n";
	int choice1;
	cin >> choice1;
	if (choice1 == 1) {
		for (int i = 0; i < numTrains - 1; i++) {
			for (int j = 0; j < numTrains - i - 1; j++) {
				if (trains[j].ticket_price > trains[j + 1].ticket_price) {
					Train temp = trains[j];
					trains[j] = trains[j + 1];
					trains[j + 1] = temp;
				}
			}
		}

	}
	if (choice1 == 2) {
		for (int i = 0; i < numTrains - 1; i++) {
			for (int j = 0; j < numTrains - i - 1; j++) {
				if (trains[j].ticket_price < trains[j + 1].ticket_price) {
```

```
                                        Train temp = trains[j];
                                        trains[j] = trains[j + 1];
                                        trains[j + 1] = temp;
                                    }
                                }
                            }
                        }
                        break;
                    }
                    case 2: {
                        cout << "Выберите тип сортировки:\n"
                            << "1. По возрастанию \n"
                            << "2. По убыванию \n";
                        int choice1;
                        cin >> choice1;
                        if (choice1 == 1) {
                            for (int i = 0; i < numTrains - 1; i++) {
                                for (int j = 0; j < numTrains - i - 1; j++) {
                                    if (trains[j].free_spaces> trains[j + 1].free_spaces) {
                                        Train temp = trains[j];
                                        trains[j] = trains[j + 1];
                                        trains[j + 1] = temp;
                                    }
                                }
                            }

                        }
                        if (choice1 == 2) {
                            for (int i = 0; i < numTrains - 1; i++) {
                                for (int j = 0; j < numTrains - i - 1; j++) {
                                    if (trains[j].free_spaces < trains[j + 1].free_spaces) {
                                        // Перестановка элементов в порядке убывания мест
                                        Train temp = trains[j];
                                        trains[j] = trains[j + 1];
                                        trains[j + 1] = temp;
```

```
                }
            }
        }
    }
    break;
}
case 3:{
                cout << "Выберите тип сортировки:\n"
            << "1. От А-Я \n"
            << "2. От Я-А \n";
        int choice2;
        cin >> choice2;
        if (choice2 == 1) {
            for (int i = 0; i < numTrains - 1; i++) {
                for (int j = 0; j < numTrains - i - 1; j++) {
                    if (strcmp(trains[j].company, trains[j + 1].company) >
0) {
                        Train temp = trains[j];
                        trains[j] = trains[j + 1];
                        trains[j + 1] = temp;
                    }
                }
            }
        }
        if (choice2 == 2) {
            for (int i = 0; i < numTrains - 1; i++) {
                for (int j = 0; j < numTrains - i - 1; j++) {
                    if (strcmp(trains[j].company, trains[j + 1].company) <
0) {
                        Train temp = trains[j];
                        trains[j] = trains[j + 1];
                        trains[j + 1] = temp;
                    }
                }
            }
```

```
                }
                break;
        }
case 4: {
        cout << "Выберите тип сортировки:\n"
                << "1. От А-Я \n"
                << "2. От Я-А \n";
        int choice2;
        cin >> choice2;
        if (choice2 == 1) {
                for (int i = 0; i < numTrains - 1; i++) {
                        for (int j = 0; j < numTrains - i - 1; j++) {
                                if (strcmp(trains[j].destination, trains[j + 1].destination) > 0) {
                                        Train temp = trains[j];
                                        trains[j] = trains[j + 1];
                                        trains[j + 1] = temp;
                                }
                        }
                }
        }
        if(choice2==2){
        for (int i = 0; i < numTrains - 1; i++) {
                for (int j = 0; j < numTrains - i - 1; j++) {
                        if (strcmp(trains[j].destination, trains[j + 1].destination) < 0) {
                                Train temp = trains[j];
                                trains[j] = trains[j + 1];
                                trains[j + 1] = temp;
                        }
                }
        }
}
        break;
}
case 5: {
        cout << "Выберите тип сортировки:\n"
```

```
                << "1. По возрастанию \n"
                << "2. По убыванию \n";
            int choice1;
            cin >> choice1;
            if (choice1 == 1) {
                for (int i = 0; i < numTrains - 1; i++) {
                    for (int j = 0; j < numTrains - i - 1; j++) {
                        if (trains[j].code > trains[j + 1].code) {
                            // Перестановка элементов в порядке возрастания
мест
                            Train temp = trains[j];
                            trains[j] = trains[j + 1];
                            trains[j + 1] = temp;
                        }
                    }
                }

            }
            if (choice1 == 2) {
                for (int i = 0; i < numTrains - 1; i++) {
                    for (int j = 0; j < numTrains - i - 1; j++) {
                        if (trains[j].code < trains[j + 1].code) {
                            // Перестановка элементов в порядке убывания мест
                            Train temp = trains[j];
                            trains[j] = trains[j + 1];
                            trains[j + 1] = temp;
                        }
                    }
                }
            }
            break;
        }

        default:
            cout << "Некорректный выбор. Сортировка не выполнена." << endl;
```

```
            break;
        }
}
void searchRecord(const Train trains[], int numTrains) {
        if (numTrains == 0) {
            printf("Нет доступных записей.\n");
            return;
        }
        char searchsmt[50];
        printf("Введите компанию или пункт назначения для поиска: ");
        scanf_s("%49s", searchsmt, sizeof(searchsmt));
        printf("По вашему запросу найдено:\n");
        printf("|==========================================================
========================================|\n");
        printf("| Id |       Компания       |                Пункт назначения                | Цена билета |
Свободных мест|\n");
        printf("|==========================================================
========================================|\n");
        bool found = false;
        for (int i = 0; i < numTrains; ++i) {
            if (strcmp(trains[i].company, searchsmt) == 0 || strcmp(trains[i].destination,
searchsmt) == 0) {
                printf("| %-3d| %-18.18s | %-42.42s | %-11.2f | %-6d       | \n",
                    trains[i].code,          trains[i].company,          trains[i].destination,
trains[i].ticket_price, trains[i].free_spaces);
                found = true;
            }

        }
        printf("|==========================================================
========================================|\n");
        if (!found) {
            cout << "Запись с указанным названием не найдена." << endl;
        }
}
```

```
double findAverageTicketPrice(const Train trains[], int numTrains) { //среднее
	if (numTrains == 0) {
		cout << "Нет доступных записей." << endl;
		return 0.0;
	}

	double total = 0.0;
	for (int i = 0; i < numTrains; i++) {
		total += trains[i].ticket_price;
	}

	return total / numTrains;
}
void findTrainsWithMaxTicketPrice(const Train trains[], int numTrains) {
	if (numTrains == 0) {
		printf("Нет доступных записей.\n");
		return;
	}

	float maxTicketPrice = trains[0].ticket_price;

	for (int i = 1; i < numTrains; i++) {
		if (trains[i].ticket_price > maxTicketPrice) {
			maxTicketPrice = trains[i].ticket_price;
		}
	}
	cout << "Рейс с самой высокой ценой: " << endl;
	printf("|==========================================================
=========================================|\n");
	printf("| Id |	Компания	|		Пункт назначения		| Цена билета |
Свободных мест|\n");
	printf("|==========================================================
=========================================|\n");
	for (int i = 0; i < numTrains; i++) {
		if (trains[i].ticket_price == maxTicketPrice) {
```

```
                printf("| %-3d| %-18.18s | %-42.42s | %-11.2f | %-6d        | \n", trains[i].code,
trains[i].company, trains[i].destination, trains[i].ticket_price, trains[i].free_spaces);
            }
        }
        printf("|=============================================================
=======================================|\n");
}
```

ПРИЛОЖЕНИЕ Б

ИНТЕРФЕЙС ПРОГРАММЫ

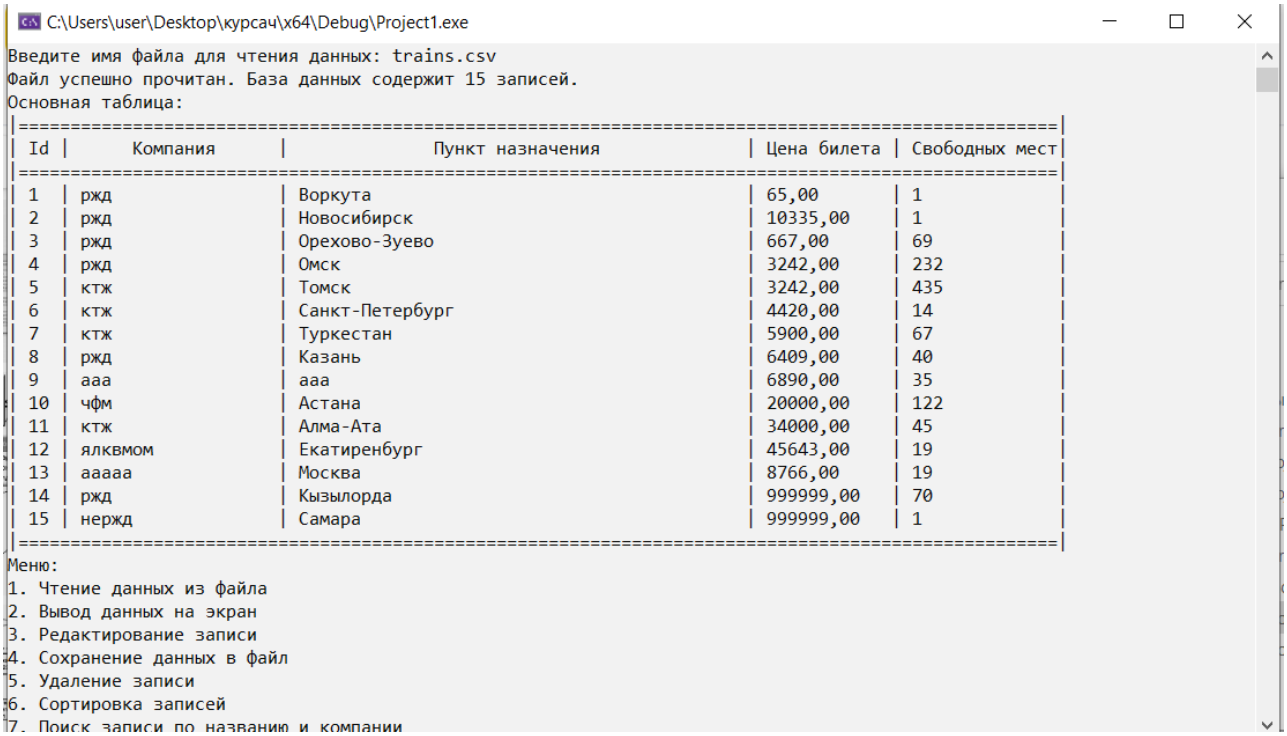

Рисунок 1 – Пример скриншота консоли

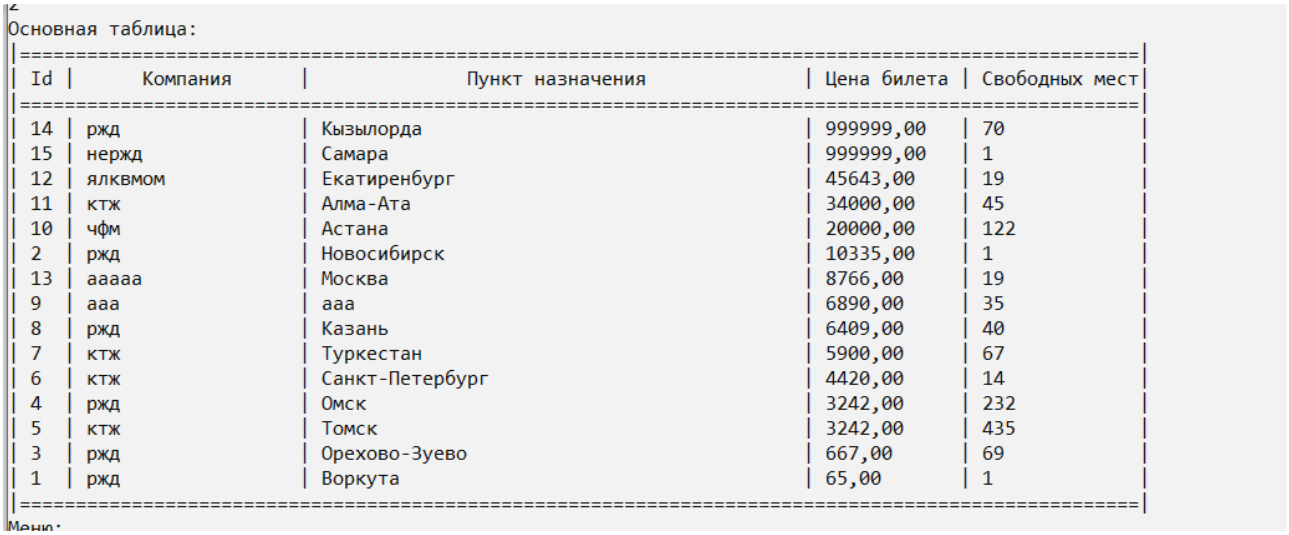

Рисунок 2 – Пример работы функции сортировка

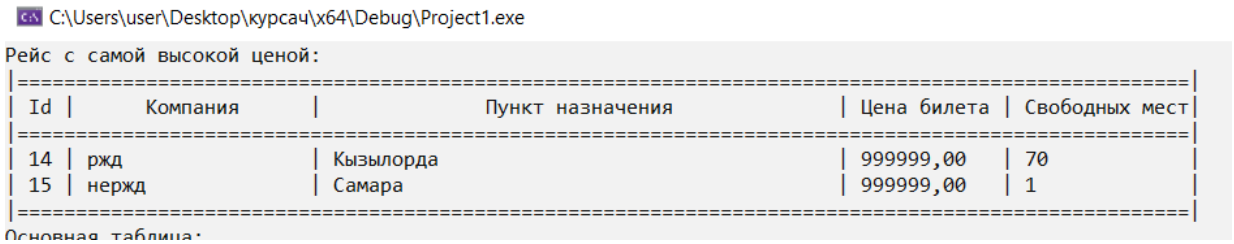

Рисунок 3 – Пример работы функции для поиска максимальной цены