Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра экономической информатики

Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

Программа управления расписания поездов

БГУИР КП 1-40 05 01-09 004 ПЗ

Студент: гр. 724402 Чернявский Я.А.

Руководитель: Сторожев Д.А.

Минск 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
1. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ. 5
1.1 Уточнение цели разрабатываемого приложения. 5
1.2 Разработка функциональных требований. 5
1.4 Технологии и язык, используемые в курсовом проекте. 5
1.4.1 Язык и среда разработки. 5
2. СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ 6
3. ОПИСАНИЕ КЛАССОВ 8
4. ДИАГРАММА КЛАССОВ 13
5. РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 14
ЗАКЛЮЧЕНИЕ 16
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 17
ПРИЛОЖЕНИЕ А 18
ПРИЛОЖЕНИЕ Б 19
ПРИЛОЖЕНИЕ В 20
ПРИЛОЖЕНИЕ Г 21

ВВЕДЕНИЕ

В наше время сложно представить повседневную жизнь без электронных девайсов. Ежедневно мы используем десятки разных электроприборов, разные виды ЭВМ вроде смартфонов, ноутбуков, ПК, фитнес-браслетов и умных часов. Каждый день на рынке гаджетов появляются различные новинки с всё более совершенными и продвинутыми встроенными технологиями.

Все устройства, которые мы используем на сегодняшний день, работают именно благодаря программированию – от смартфонов до электрических чайников. Для всех устройств разрабатывается огромное количество программ и алгоритмов работы. Чем лучше написан код ПО устройства, тем эффективнее оно работает – меньше количество ошибок, затраты энергии, времени, ресурсов. Также в сегодняшних гаджетах используются элементы машинного обучения, которые теперь становятся не экспериментами, а сегодняшней реальностью.

Но прогресс не стоит на месте, поэтому люди постоянно ищут способ как-либо уменьшить объём своей работы, перекинув их на гаджеты.

На сегодняшний день почти все жители Беларуси имеют доступ в интернет. Благодаря этому появилась возможность общаться, не используя обычную голосовую мобильную связь. Но это не единственный путь использования интернета. Эту связь можно использовать для заказа транспорта, продуктов, билетов и т.д. и т.п. С помощью баз данных интернет-магазины могут хранить данные о сотнях и тысячах заказов, сделанных через формы в интернете. Также их можно использовать для хранения данных о рейсах общественного транспорта, в т.ч. поездов. Благодаря простоте интерфейса, использовании базы данных рейсов и записи в файл мы запросто можем посмотреть информацию о рейсе, его дате, направлении или свободных местах, находящихся в полях записи.

1. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ.

1.1 Уточнение цели разрабатываемого приложения.

Цель:
Разработать удобное приложение для работы с расписанием поездов.

1.2 Разработка функциональных требований.

После проведения анализа вышеописанных требований и целей, для которых идёт создание ПС, были разработаны следующие функциональные требования:

- Хранение информации;
- Добавление, удаление и редактирование информации по определенным рейсам;
- Сортировка, поиск рейсов;
- Сохранение информации в файл, а также возможность продолжить процесс;

1.4 Технологии и язык, используемые в курсовом проекте.

1.4.1 Язык и среда разработки.

Так как я собираюсь писать приложение на платформе Windows, был выбран язык программирования C++.

C++ — компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его

применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (плееров, игр).

Для создания приложений на платформе Windows корпорация Microsoft выпустила специальную среду разработки: Visual Studio. Для реализации курсового проекта используется самая свежая версия этой среды. Она упрощает создание пользовательского интерфейса, сводит к минимуму количество необходимых операций для компиляции и запуска проекта, а также предоставляет огромное количество ресурсов для улучшения качества кода программиста.

2. СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ

Для реализации поставленной задачи используются текстовые файлы. В текстовых файлах хранится информация о поездах различных классов: грузовые поезда, поезда с местами повышенной комфортности, поезда на электрической тяге.

Данные о дизельных поездах хранятся в файле Diesel.txt. Данные о электричках хранятся в файле Electro.txt. Данные о поездах-выставках хранятся в файле exhTrain.txt. Данные о скоростных поездах хранятся в файле SpeedTrain.txt. Данные грузовых поездах хранятся в файле carryWeight.txt. Данные о сверхдлинных поездах хранятся в файле carryWood.txt.

Промежуточные данные хранятся в шаблонном классе List.

Таблица 1 - Diesel.txt

Направление, string	Свободные места, string	Дата, int	Стоимость билета, string (int)	Время отправления, int
Осиповичи	15	13	25	15
Белая	22	16	22	20
….	….	….		

Таблица 2 - Electro.txt

Направление, string	Свободные места, string	Цена, int	Дата, int	Время, int	Время между

					остановкам и, int
Барановичи	13	22	9	15	7
Пустоши	20	27	7	11	4
….	….	….	….		….

Таблица 3 - exhTrain.txt

Направление, string	Свободные места, string	Цена, int	Дата, int	Время, int	Описание выставки, string
Мир	2	40	22	15	Музей
….	….	….	….	….	….

Таблица 4 - SpeedTrain.txt

Направление, string	Свободные места, string	Цена, int	Дата, int	Время, int	Максимально развиваемая скорость
Москва	22	50	24	7	450
….	….	….	….	….	

Таблица 5 - carryWeight.txt

Направление, string	Свободные места, string	Цена, int	Дата, int	Время, int	К-во вагонов с грузом, int
Череповец	0	250	8	15	30
….	….	….	….	….	

Таблица 6 - carryWood.txt

Направление, string	Свободные места, string	Цена, int	Дата, int	Время, int	Длина состава, int
Вроцлав	0	120	6	6	1250
….	….	….	….	….	

3. ОПИСАНИЕ КЛАССОВ

Базовый класс:

```
class Item
{
protected:
	string nameItem;			//Направление
	string seats;			//Свободные места
	int price;			//Стоимость билета
	int date;			//Дата отправления
public:
	Item();				//Конструктор без параметров
	Item(const Item&);			//Конструктор копирования
	Item(string, string, int, int);	//Конструтор с параметрами
	~Item();				//Деструктор
	void setData();			//Метод, определяющий поля элемента
	void showData();			//Метод, выводящий данные полей элемента
	void editData();			//Метод, изменяющий данные полей элемента
	void showLabel();			//Метод, выводящий полей элемента
	virtual string getName();			//Метод, возвращающий nameItem
	virtual string getType();			//Метод, возвращающий producer
	friend ostream& operator<<(ostream&, Item&);		//Перегрузка вывода
	friend istream& operator >> (istream&, Item&);		//Перегрузка ввода
	friend ofstream& operator<<(ofstream&, Item&);		//Перегрузка вывода
	friend ifstream& operator >> (ifstream&, Item&);		//Перегрузка ввода
	Item operator=(const Item&);			//Перегрузка оператора присваивания
	bool operator==(const Item&);			//Перегрузка оператора сравнения
	void showHeader();				//Метод, показывающий заголовки элемента
	friend bool compareName(const Item *, const  Item *,int); //Метод,сравнивающий поле nameItem
	friend bool compareType(const Item *, const  Item *,int); //Метод,сравнивающий поле seats
	friend bool comparePrice(const Item *, const  Item *,int); //Метод,сравнивающий поле price
	friend bool compareDate(const Item *, const  Item *,int);//Метод,сравнивающий поле date
};
```

Класс, содержащий данные о поездах:

```
class Trains : public Item
{
protected:
	int hours;			//Время в пути
public:
	Trains();				//Конструктор без параметров
	Trains(const Trains&);		//Конструктор копирования
	Trains(string,string,int,int,int);	//Конструктор с параметрами
```

```
        ~Trains();                          //Деструктор
        void setData();                     //Метод, определяющий поля элемента
        void showData();                    //Метод, выводящий данные поля элемента
        void editData();                    //Метод, изменяющий данные поля элемента
        void showLabel();                   //Метод, выводящий поля элемента
        friend ostream& operator<<(ostream&, Trains&);      //Перегрузка оператора вывода
        friend istream& operator >> (istream&, Trains&);    //Перегразка оператора ввода
        Trains operator=(const Trains&);                    //Перегрузка оператора присваивания
        bool operator==(const Trains&);                     //Перегрузка оператора сравнения
        friend ofstream& operator<<(ofstream&, Trains&);    //Перегрузка оператора вывода
        friend ifstream& operator >> (ifstream&, Trains&);  //Перегрузка оператора ввода
        void showHeader();                  //Метод, выводящий заголовки элемента
        friend bool compareTime(const Trains *, const Trains *, int);      //Метод, сравнивающий поле hours
```

Класс, содержащий данные о электричках:

```
class Electro: public Trains
{
protected:
        string waypointTime;                //Время в пути
public:
        Electro();                      //Конструктор без параметров
        Electro(const Electro&);        //Конструктор копирования
        Electro(string, string, int, int , string);             //Конструктор с параметрами
        ~Cartoon();                                             //Деструктор
        void setData();                 //Метод, определяющий поля элемента
        void showData();                //Метод, выводящий данные поля элемента
        void showLabel();               //Метод, выводящий поля элемента
        friend ostream& operator<<(ostream&, Electro&);             //Перегрузка вывода
        friend istream& operator >> (istream&, Electro&);           //Перегрузка ввода
        friend ofstream& operator<<(ofstream&, Electro&);           //Перегрузка вывода
        friend ifstream& operator >> (ifstream&, Electro&);         //Перегрузка ввода
        Electro operator=(const Electro&);                  //Перегрузка оператора присваивания
        bool operator==(const Electro&);                    //Перегрузка оператора сравнения
        friend bool comparePoint(const Electro *, const Electro *, int);  //Метод, сравнивающий поле waypointTime
        void showHeader();              //Метод, показывающий заголовки элемента
        void editData();                //Метод, изменяющий данные поля элемента

};
```

Класс, содержащий данные о дизельных поездах с коротким маршрутом:

```
class Diesel : public Trains
{
protected:
        string additionalStops;             //Количество остановок по пути следования (для определения, пропускаются
ли необязательные остановки)
public:
        Diesel();                       //Конструктор без параметров
        Diesel(const Diesel&);          //Конструктор копирования
        Diesel(string, string, int, int, int , string);             //Конструктор с параметрами
        ~Diesel();                  //Деструктор
        void setData();             //Метод, определяющий данные поля элемента
        void showData();            //Метод, выводящий данные поля элемента
        void setOptional();         //Метод заполнения определенных данных
        void editData();            //Метод, изменяющий данные поля элемента
        void showLabel();           //Метод, показывающий поля элемента
        friend ostream& operator<<(ostream&, Diesel&);              //Перегрузка вывода
```

```
        friend istream& operator >> (istream&, Diesel&);              //Перегрузка ввода
        friend ofstream& operator<<(ofstream&, Diesel&);               //Перегрузка вывода
        friend ifstream& operator >> (ifstream&, Diesel&);            //Перегрузка ввода
        Diesel operator=(const Diesel&);          //Перегрузка оператора присваивания элемента
        bool operator==(const Diesel&);              //Перегрузка оператора сравнения
        void showHeader();                          //Метод, выводящий заголовки поля элемента
        friend bool compareStops( Diesel *, Diesel *, int);            //Метод, сравнивающий поля additionalStops
};
```

Класс, содержащий данные о особых поездах для дальних переездов:

```
class vipTrain : public Item
{
protected:
        string vipCount;             //количество мест в VIP вагонах
public:
        vipTrain();                  //Конструктор без параметров
        vipTrain(const vipTrain&);     //Конструктор копирования
        vipTrain(string, string, int, int, int, string);             //Конструктор с параметрами
        ~ vipTrain();               //Деструктор
        void setData();              //Метод, определяющий данные поля элемента
        void showData();               //Метод, выводящий данные поля элемента
        void showLabel();              //Метод, выводящий поля элемента
        void setOptional();           //Метод заполнения определенных данных
        friend ostream& operator<<(ostream&, vipTrain&);               //Перегрузка вывода
        friend istream& operator >> (istream&, vipTrain&);             //Перегрузка ввода
        friend ofstream& operator<<(ofstream&, vipTrain&);              //Перегрузка вывода
        friend ifstream& operator >> (ifstream&, vipTrain&);          //Перегрузка ввода
        vipTrain operator=(const vipTrain&);             //Перегрузка оператора присваивания
        bool operator==(const vipTrain&);               //Перегрузка оператора сравнения
        friend bool compareVipSeats( vipTrain *,  vipTrain *, int);              //Метод, сравнивающий поле vipCount
        void showHeader();            //Метод, выводящий заголовки поля элемента
        void editData();             //Метод, изменяющий данные поля элемента
};
```

Класс, содержащий данные о поездах-выставках:

```
class exhTrain : public vipTrain
{
protected:
        string descExh;             //описание выставки
public:
        exhTrain();                          //Конструктор без параметров
        exhTrain(const exhTrain&);           //Конструктор копирования
        exhTrain(string, string, int, int, string, string);         //Конструктор с параметрами
        ~exhTrain();               //Деструктор
        void setData();              //Метод, изменяющий данные поля элемента
        void showData();               //Метод, выводящий данные поля элемента
        void showLabel();              //Метод, выводязий поля элемента
        void setOptional();           //Метод заполнения определенных данных
        friend ostream& operator<<(ostream&, exhTrain&);               //Перегрузка вывода
        friend istream& operator >> (istream&, exhTrain&);             //Перегрузка ввода
        friend ofstream& operator<<(ofstream&, exhTrain&);              //Перегрузка вывода
        friend ifstream& operator >> (ifstream&, exhTrain&);          //Перегрузка ввода
        exhTrain operator=(const exhTrain&);                 //Перегрузка оператора присваивания
        bool operator==(const exhTrain&);                   //Перегрузка оператора сравнения
        friend bool compareTheme(exhTrain *, exhTrain *, int);     //Метод, сравнивающий поле descExh
        void showHeader();            //Метод, выводящий заголовки поля элемента
        void editData();             //Метод, изменяющий данные поля элемента};
```

Класс, содержащий данные о грузовых поездах:

```
class carryWeight : public Item
{
protected:
	int amountLoaded;		//количество вагонов с грузом
public:
	carryWeight();		//Конструктор без параметров
	carryWeight(const carryWeight&);		//Конструктор копирования
	carryWeight(string, string, int, int, string, int);		//Конструктор с параметрами
	~carryWeight();		//Деструктор
	void setData();		//Метод, определяющий данные поля элемента
	void showData();		//Метод, выводящий данные поля элемента
	void showLabel();		//Метод, выводящий поля элемента
	void setOptional();		//Метод заполнения определенных данных
	friend ostream& operator<<(ostream&, carryWeight&);		//Перегрузка вывода
	friend istream& operator >> (istream&, carryWeight&);		//Перегрузка ввода
	friend ofstream& operator<<(ofstream&, carryWeight&);		//Перегрузка вывода
	friend ifstream& operator >> (ifstream&, carryWeight&);		//Перегрузка ввода
	carryWeight operator=(const carryWeight&);		//Перегрузка оператора присваивания
	bool operator==(const carryWeight&);		//Перегрузка оператора сравнения
	friend bool compareStorage(carryWeight *, carryWeight *, int);		//Метод, сравнивающий поле
amountLoaded
	void showHeader();		//Метод, выводящий заголовки поля элемента
	void editData();		//Метод, изменяющий данные поля элемента

};
```

Класс, содержащий данные о сверхдлинных поездах:

```
class carryWood : public Item
{
protected:
	int avgLenght;		//Приблизительная длина состава
public:
	carryWood();		//Конструктор без параметров
	carryWood(const carryWeight&);		//Конструктор копирования
	carryWood(string, string, int, int, string, int);		//Конструктор с параметрами
	~carryWood();		//Деструктор
	void setData();		//Метод, определяющий данные поля элемента
	void showData();		//Метод, выводящий данные поля элемента
	void showLabel();		//Метод, выводящий поля элемента
	void setOptional();		//Метод заполнения определенных данных
	friend ostream& operator<<(ostream&, carryWood&);		//Перегрузка вывода
	friend istream& operator >> (istream&, carryWood&);		//Перегрузка ввода
	friend ofstream& operator<<(ofstream&, carryWood&);		//Перегрузка вывода
	friend ifstream& operator >> (ifstream&, carryWood&);		//Перегрузка ввода
	carryWood operator=(const carryWood&);		//Перегрузка оператора присваивания
	bool operator==(const carryWood&);		//Перегрузка оператора сравнения
	friend bool compareLenght(carryWeight *, carryWeight *, int);		//Метод, сравнивающий поле avgLenght
	void showHeader();		//Метод, выводящий заголовки поля элемента
	void editData();		//Метод, изменяющий данные поля элемента};
```

Класс, содержащий данные о сверхскоростных поездах:

```
class SpeedTrain : public vipTrain
{
protected:
	string maxSpeed;		//максимально развиваемая скорость
public:
	SpeedTrain();			//Конструктор без параметров
	SpeedTrain(const SpeedTrain&);
	//Конструктор копирования
	SpeedTrain(string, string, int, int, string, string);		//Конструктор с параметрами
	~SpeedTrain();			//Десктрутор
	void setData();			//Метод, определяющий данные поля элемента
	void showLabel();		//Метод, выводящий поля элемента
	void showData();		//Метод, выводящий данные поля элемента
	void setOptional();		//Метод заполнения определенных данных
	friend ostream& operator<<(ostream&, SpeedTrain&);		//Перегрузка вывода
	friend istream& operator >> (istream&, SpeedTrain&);		//Перегрузка ввода
	friend ofstream& operator<<(ofstream&, SpeedTrain&);		//Перегрузка вывода
	friend ifstream& operator >> (ifstream&, SpeedTrain&);		//Перегрузка ввода
	SpeedTrain operator=(const SpeedTrain&);		//Перегрузка оператора присваивания
	bool operator==(const SpeedTrain&);		//Перегрузка оператора сравнения
	void showHeader();		//Метод, выводящий заголовки поля элемента
	void editData();		//Метод, изменяющий данные поля элемента
	friend bool compareTypeSpeed( SpeedTrain *,  SpeedTrain *, int);		//Метод, сравнивающий поле
maxSpeed
};
```

Шаблонный класс, являющийся основой хранения промежуточных данных:

```
template<class Type>
struct Line
{
	Type obj;		//Элемент
	Line<Type> *next;	//Указатель на следующий элемент
	Line<Type> *prev;	//Указатель на предыдующий элемент
};

template<class Type>
class List
{
private:
	Line<Type> *begin;		//Указатель на начало списка
	Line<Type> *end;		//Указатель на конец списка
	int count;			//Количество элементов
public:
	List();				//Конструктор без параметров
	void push(const Type&);		//Добавить элемент в начало
	void pushBack(const Type&);	//Добавить элемент в конец
	Type dellOne();			//Удалить элемент с начала
	Type dellOneBack();		//Удалить элемент с конца
	void dellAll();			//Удалить все элементы
	int getCount();			//Метод, возвращающий поле count
	Type dellPoint(Line<Type>*);	//Удаление заданного элемента
	void show();				//Метод, выводящий все элемента списка с начала
```

```
        void showBack();                //Метод, выводящий все элемента списка с конца
        Line<Type>* operator [] (int);  //Перегрузка оператора []
        void readFile(string);          //Чтение данных из файла
        void writeFile(string);         //Запись данных в файл
        bool isEmpty();                 //Метод, проверяющий список на пустоту
        void search(const Type&);       //Поиск элемента по названию
        void sort(int, bool(*Compare)(Type*, Type*, int));      //Сортировка по всем полям
        ~List();
                //Деструктор
};
```

Структура, содержащая данные:

```
struct Line
{
        Type obj;               //Элемент
        Line<Type> *next;       //Указатель на следующий элемент
        Line<Type> *prev;       //Указатель на предыдующий элемент
};
```

Шаблонный класс интерфейс:

```
template<class Type>
class Interface
{
        Type obj;                       //Шаблонная переменная
public:
        Interface();                    //Конструктор без параметров
        Interface(const Type& obj);     //Конструктор копирования
        void mainMenu();                //Главное меню
        void orderMenu(int,string);     //Меню для работы с данными
        ~Interface();                   //Деструктор
};
```

4. ДИАГРАММА КЛАССОВ

Диаграмма классов представлена в приложении А.

5. РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

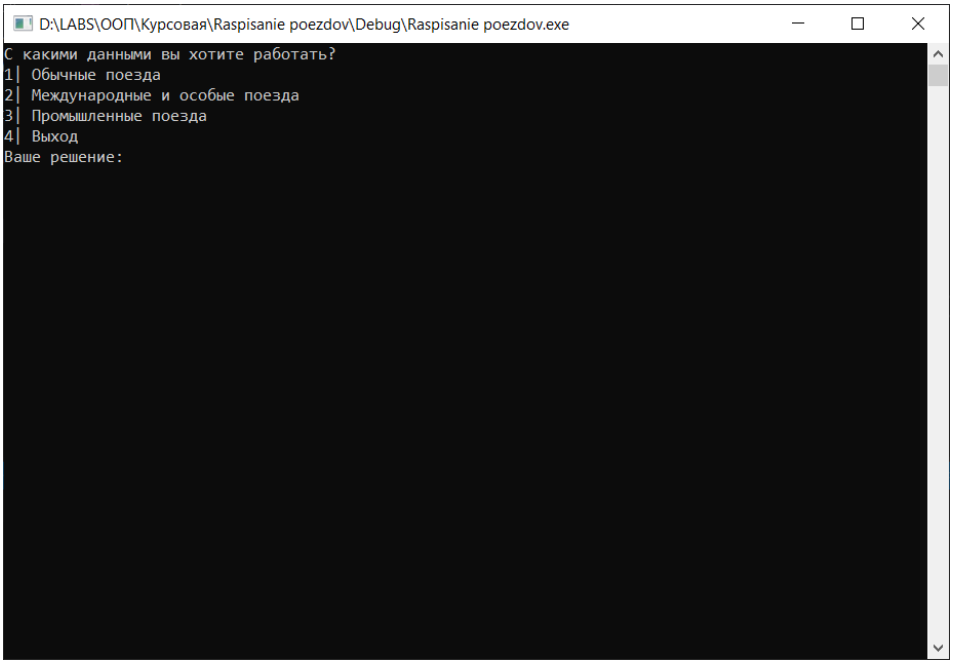


Рисунок 1 – Главное меню программы

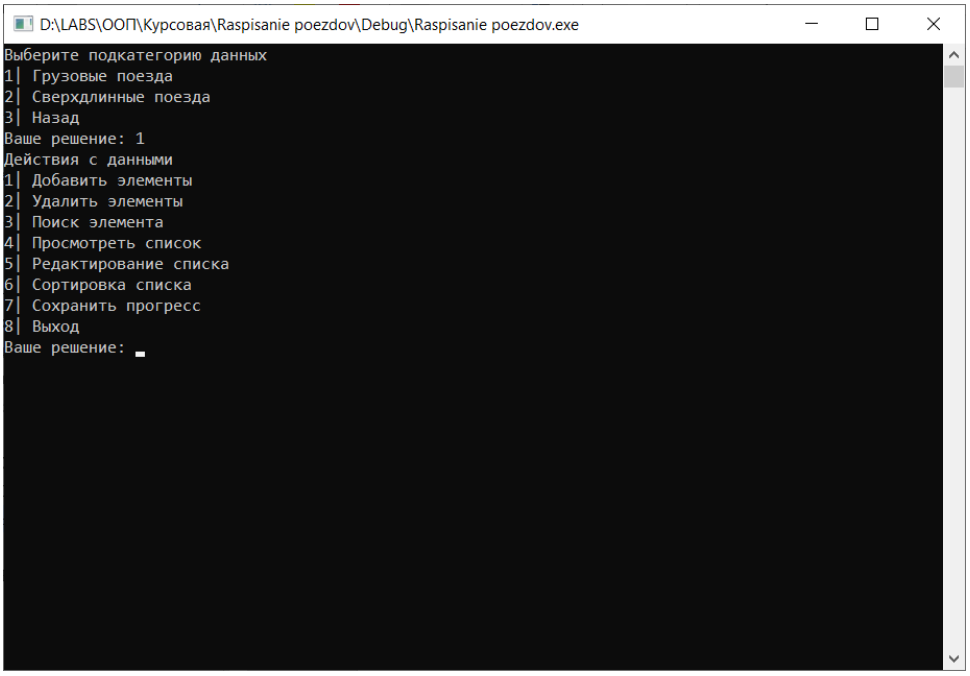


Рисунок 2 – Меню работы с данными

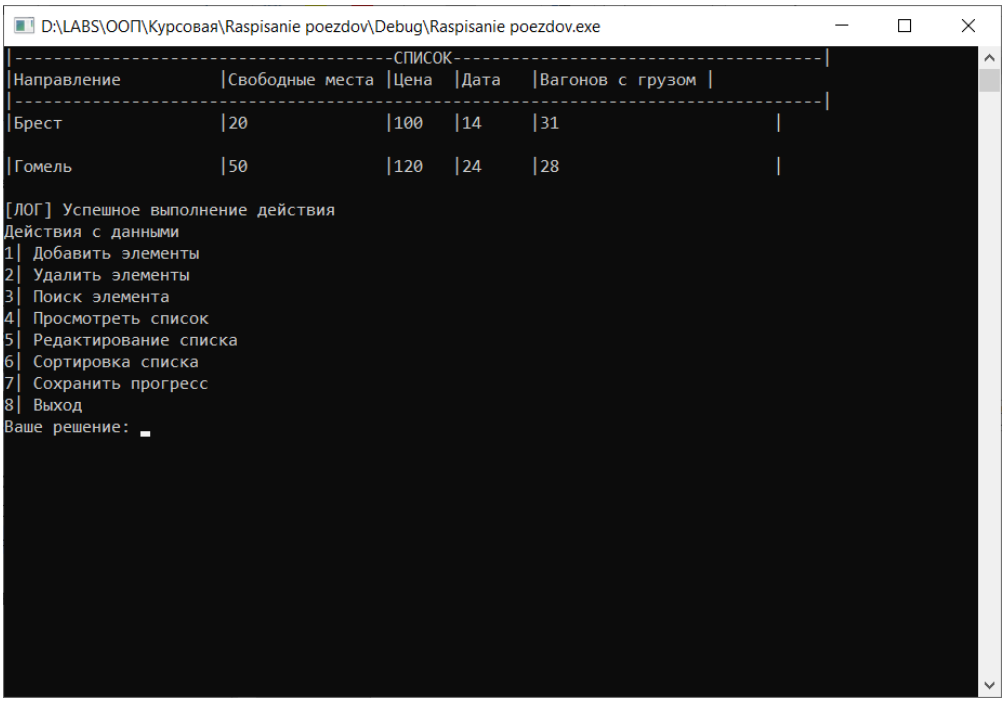


Рисунок 3 – Просмотр списка

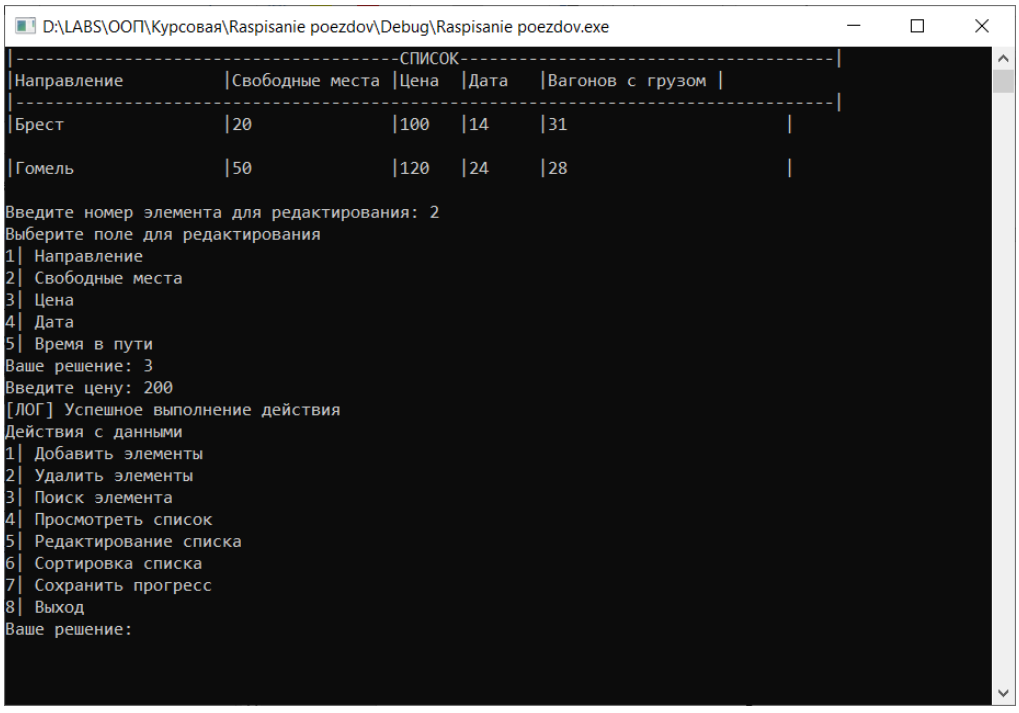


Рисунок 4 – Редактирование списка

ЗАКЛЮЧЕНИЕ

В результате работы над курсовым проектом было создано исправно работающее средство управления расписания поездов.

Разработка приложения включала в себя решения множества задач и проблем, как итог было изучено большое количество структур, методов, паттернов программирования.

Далее были изучены некоторые возможности создания приложений в Microsoft Visual Studio и формирование конкретных функциональных требований к программе на основе возможностей языка.

Были и разработаны алгоритмы работы с данными, продумана примерная архитектура приложения. Программа была отлажена и проверена.

После окончания всех этапов, перечисленных ранее в данной пояснительной записке, было создано требуемое приложение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Конструирование программ и языки программирования: Учебно-метод. пособие для студ. спец «Программное обеспечение информационных технологий»: В 2 ч. Ч. 1: Язык СИ / В.В. Бахтизин, И.М. Марина, Е.В. Шостак. – Мн.: БГУИР, 2006. – 48 с.;

[2] Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон; пер. с англ. – СПб. : ДМК, 2004. – 429 с.

[3] Дейтел, Х.М. Как программировать на C++ / Х.М. Дейтел, П.Д. Дейтел; пер. с англ. – М. : Бином, 2007. – 1152 с.

[4] Доманов, А.Т. Предварительный стандарт предприятия. Дипломные проекты(работы) : общие требования / А.Т. Доманов, Н.И. Сороко. – Мн.: БГУИР, 2009. – 171 с.

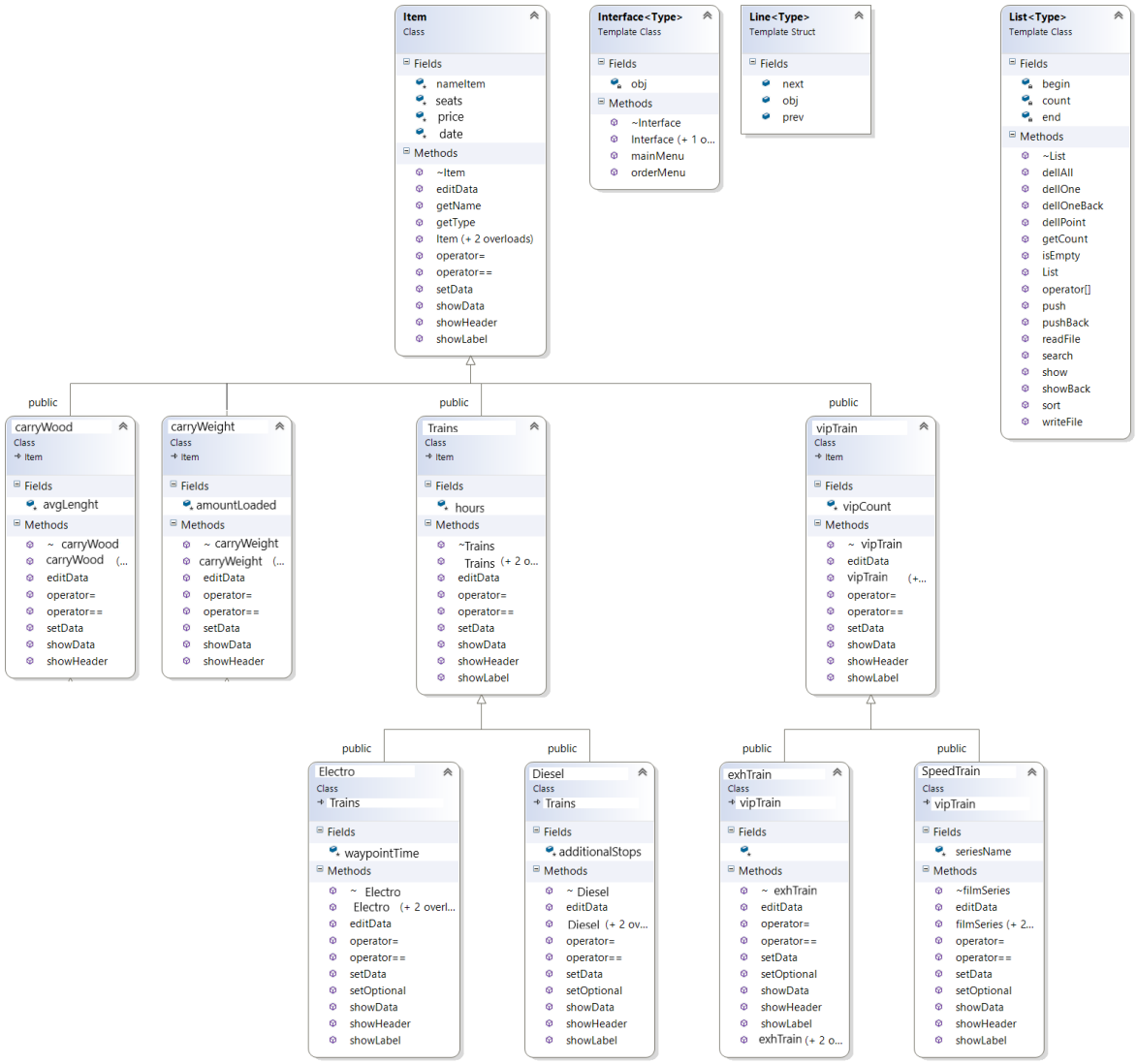
[5] Страуструп, Б. Язык программирования C++ / Б. Страуструп; специальное издание. Пер. с англ. – СПб. : BHV, 2008. – 1098 с.

[6] Скляров, В.А. Язык C++ и объектно-ориентированное программирование: справ. пособие / В.А. Скляров. – Мн. : Вышэйшая школа, 1997. – 478 с.

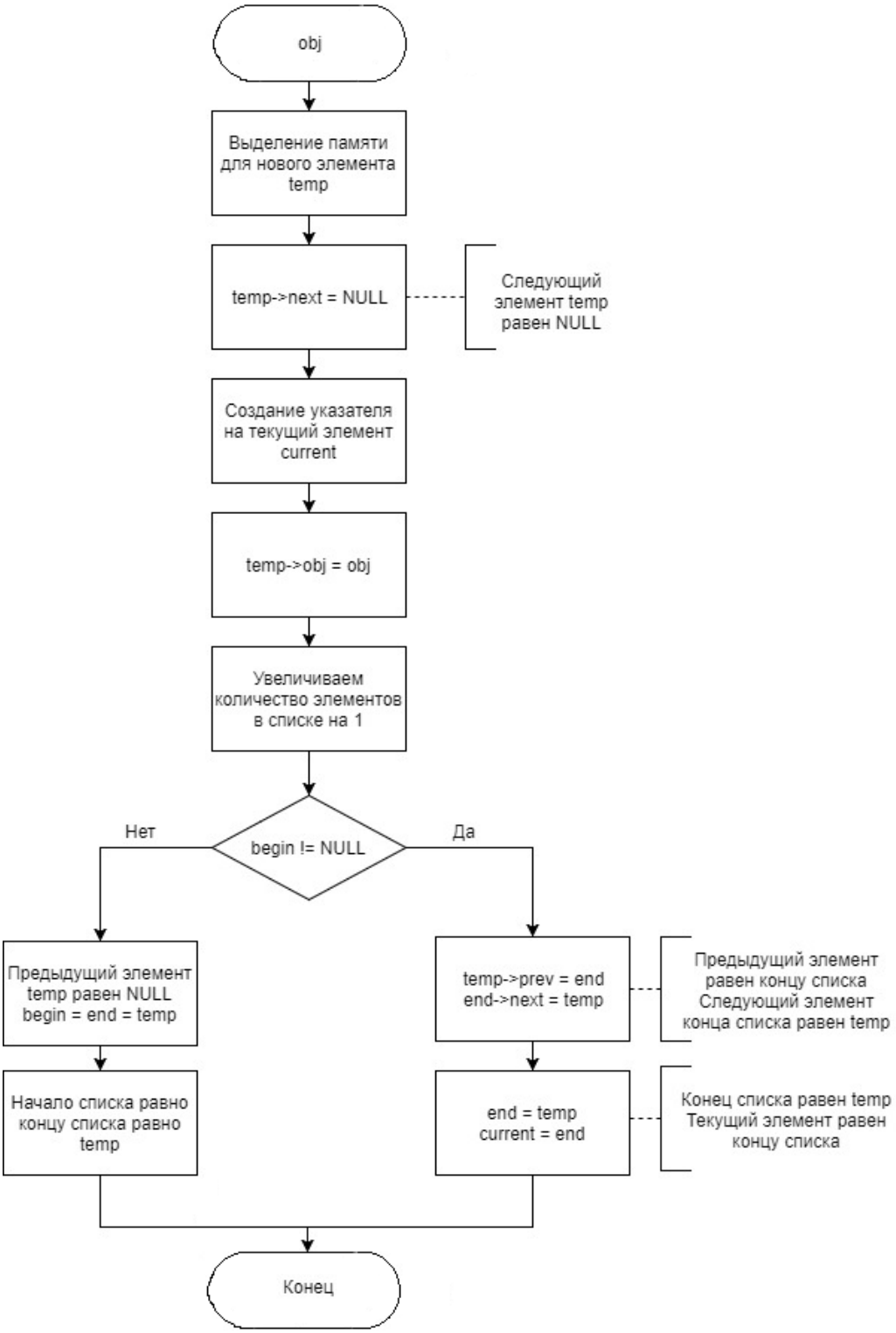
[7] Элджер, Дж. C++: библиотека программиста / Дж. Элджер. – СПб. : Питер, 2001. – с.

[8] Шилд, Г. Программирование на Borland C++ для профессионалов / Г. Шилд. – Мн. : ООО «Попури», 1998. – 800 с.

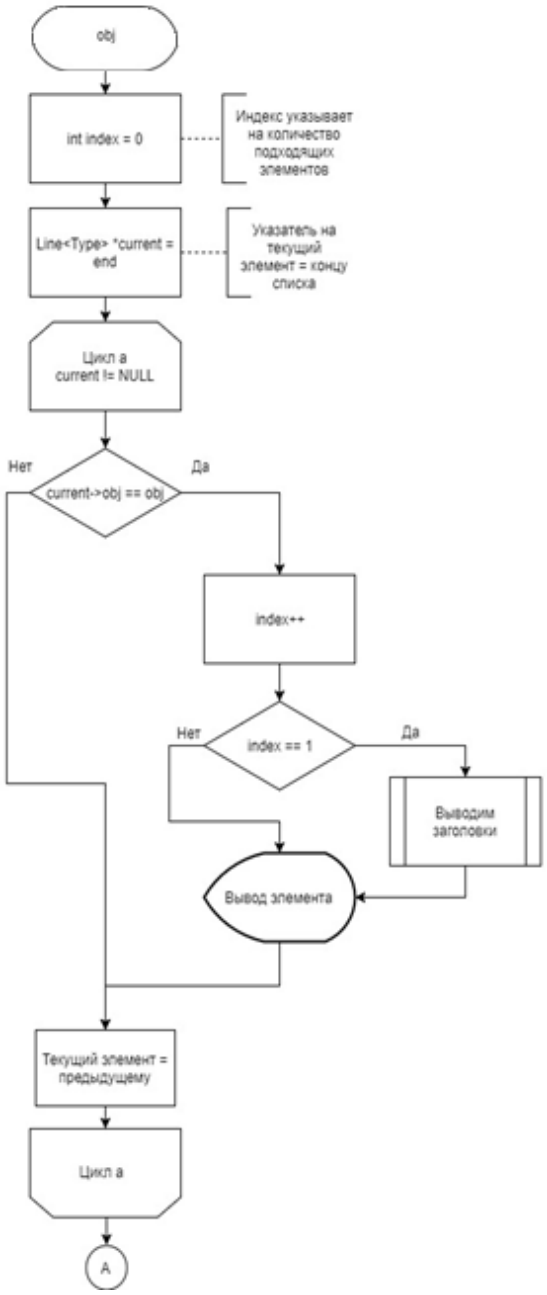
ПРИЛОЖЕНИЕ А

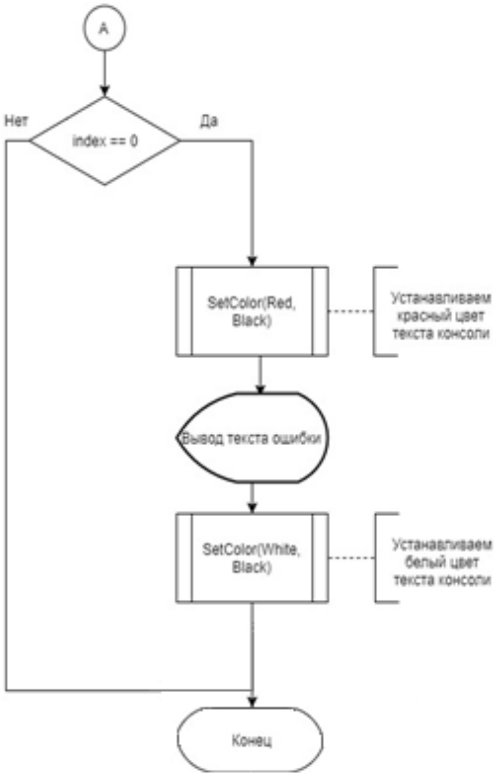


ПРИЛОЖЕНИЕ Б



ПРИЛОЖЕНИЕ В





ПРИЛОЖЕНИЕ Г
Листинг кода

Файлы заголовков:

carryWeight.h

```
#include "Item.h"

class carryWeight : public Item
{
protected:
	int amountLoaded;			//количество вагонов с грузом
public:
	carryWeight();			//Конструктор без параметров
	carryWeight(const carryWeight&);				//Конструктор
копирования
	carryWeight(string, string, int, int, int);			//Конструктор с
параметрами
	~carryWeight() = default;		//Деструктор
	void setData();				//Метод, определяющий данные поля элемента
	void showData();			//Метод, выводящий данные поля элемента
	void showLabel();			//Метод, выводящий поля элемента
	void setOptional();			//Метод заполнения определенных данных
	friend ostream& operator<<(ostream&, carryWeight&);			//Перегрузка
вывода
	friend istream& operator >> (istream&, carryWeight&);			//Перегрузка
ввода
	friend ofstream& operator<<(ofstream&, carryWeight&);			//Перегрузка
вывода
	friend ifstream& operator >> (ifstream&, carryWeight&);			//Перегрузка
ввода
	carryWeight operator=(const carryWeight&);			//Перегрузка оператора
присваивания
	bool operator==(const carryWeight&);			//Перегрузка оператора
сравнения
	friend bool compareTimeStorage(const carryWeight*, const carryWeight*, int);
//Метод, сравнивающий поле amountLoaded
	void showHeader();			//Метод, выводящий заголовки поля элемента
	void editData();			//Метод, изменяющий данные поля элемента
};
```

carryWood.h

```
#include "Item.h"

class carryWood : public Item
{
protected:
	int amountLenght;			//Общая длина вагонов
public:
	carryWood();			//Конструктор без параметров
	carryWood(const carryWood&);				//Конструктор копирования
	carryWood(string, string, int, int, int);			//Конструктор с параметрами
	~carryWood() = default;		//Деструктор
	void setData();				//Метод, определяющий данные поля элемента
```

```
        void showData();                    //Метод, выводящий данные поля элемента
        void showLabel();                   //Метод, выводящий поля элемента
        void setOptional();                 //Метод заполнения определенных данных
        friend ostream& operator<<(ostream&, carryWood&);                 //Перегрузка
вывода
        friend istream& operator >> (istream&, carryWood&);               //Перегрузка ввода
        friend ofstream& operator<<(ofstream&, carryWood&);               //Перегрузка
вывода
        friend ifstream& operator >> (ifstream&, carryWood&);             //Перегрузка ввода
        carryWood operator=(const carryWood&);              //Перегрузка оператора
присваивания
        bool operator==(const carryWood&);                  //Перегрузка оператора сравнения
        friend bool compareLong(const carryWood*, const carryWood*, int);
//Метод, сравнивающий поле amountLenght
        void showHeader();                //Метод, выводящий заголовки поля элемента
        void editData();                  //Метод, изменяющий данные поля элемента
};
```

Diesel.h

```
#include "Trains.h"

class Diesel : public Trains
{
protected:
        string addStops;                //Количество остановок по пути следования (для
определения, пропускаются ли необязательные остановки)
public:
        Diesel();                           //Конструктор без параметров
        Diesel(const Diesel&);           //Конструктор копирования
        Diesel(string, string, int, int, int, string);               //Конструктор с
параметрами
        ~Diesel();                  //Деструктор
        void setData();                //Метод, определяющий данные поля элемента
        void showData();               //Метод, выводящий данные поля элемента
        void setOptional();            //Метод заполнения определенных данных
        void editData();               //Метод, изменяющий данные поля элемента
        void showLabel();              //Метод, показывающий поля элемента
        friend ostream& operator<<(ostream&, Diesel&);                  //Перегрузка вывода
        friend istream& operator >> (istream&, Diesel&);                //Перегрузка ввода
        friend ofstream& operator<<(ofstream&, Diesel&);                //Перегрузка вывода
        friend ifstream& operator >> (ifstream&, Diesel&);              //Перегрузка ввода
        Diesel operator=(const Diesel&);           //Перегрузка оператора присваивания
элемента
        bool operator==(const Diesel&);               //Перегрузка оператора сравнения
        void showHeader();                               //Метод, выводящий заголовки поля
элемента
        friend bool compareStops(const Diesel*, const Diesel*, int);               //Метод,
сравнивающий поля additionalStops
};
```

Electro.h

```
#include "Trains.h"

class Electro : public Trains
{
protected:
	int waypointTime;                              //Время между остановками
public:
	Electro();                                  //Конструктор без параметров
	Electro(const Electro&);              //Конструктор копирования
	Electro(string, string, int, int, int, int);         //Конструктор с параметрами
	~Electro();                                                 //Деструктор
	void setData();                 //Метод, определяющий поля элемента
	void showData();                //Метод, выводящий данные поля элемента
	void showLabel();               //Метод, выводящий поля элемента
	void setOptional();
	friend ostream& operator<<(ostream&, Electro&);               //Перегрузка вывода
	friend istream& operator >> (istream&, Electro&);             //Перегрузка ввода
	friend ofstream& operator<<(ofstream&, Electro&);             //Перегрузка вывода
	friend ifstream& operator >> (ifstream&, Electro&);           //Перегрузка ввода
	Electro operator=(const Electro&);           //Перегрузка оператора присваивания
	bool operator==(const Electro&);             //Перегрузка оператора сравнения
	friend bool compareWpTime(const Electro*, const Electro*, int);
//Метод, сравнивающий поле waypointTime
	void showHeader();                //Метод, показывающий заголовки элемента
	void editData();                  //Метод, изменяющий данные поля элемента
};
```

exhTrain.h

```
#include "vipTrain.h"

class exhTrain : public vipTrain
{
protected:
	string descExh;               //описание выставки
public:
	exhTrain();                                 //Конструктор без параметров
	exhTrain(const exhTrain&);             //Конструктор копирования
	exhTrain(string, string, int, int, string, string);          //Конструктор с
параметрами
	~exhTrain();                //Деструктор
	void setData();                //Метод, изменяющий данные поля элемента
	void showData();               //Метод, выводящий данные поля элемента
	void showLabel();              //Метод, выводязий поля элемента
	void setOptional();            //Метод заполнения определенных данных
	friend ostream& operator<<(ostream&, exhTrain&);                  //Перегрузка вывода
	friend istream& operator >> (istream&, exhTrain&);                //Перегрузка ввода
	friend ofstream& operator<<(ofstream&, exhTrain&);                //Перегрузка вывода
	friend ifstream& operator >> (ifstream&, exhTrain&);              //Перегрузка ввода
	exhTrain operator=(const exhTrain&);                     //Перегрузка оператора
присваивания
	bool operator==(const exhTrain&);                        //Перегрузка оператора
сравнения
	friend bool comparedescExh(const exhTrain*, const exhTrain*, int);     //Метод,
сравнивающий поле descExh
	void showHeader();             //Метод, выводящий заголовки поля элемента
```

```
	void editData();		//Метод, изменяющий данные поля элемента
};
```

header.h

```
#include <iostream>
#include <iomanip>
#include <string>
#include <fstream>
#include <Windows.h>
using namespace std;


bool isFigure(string text);
/*  Шаблонные функции приведения типа  */
template<class Type> bool compareName(Type* t1, Type* t2, int ord)
{
	return compareName(dynamic_cast<const Item*>(t1), dynamic_cast<const Item*>(t2),
ord);
}
template<class Type> bool compareType(Type* t1, Type* t2, int ord)
{
	return compareType(dynamic_cast<const Item*>(t1), dynamic_cast<const Item*>(t2),
ord);
}
template<class Type> bool comparePrice(Type* t1, Type* t2, int ord)
{
	return comparePrice(dynamic_cast<const Item*>(t1), dynamic_cast<const Item*>(t2),
ord);
}
template<class Type> bool compareDate(Type* t1, Type* t2, int ord)
{
	return compareDate(dynamic_cast<const Item*>(t1), dynamic_cast<const Item*>(t2),
ord);
}
template<class Type> bool compareTime(Type* t1, Type* t2, int ord)
{
	return compareTime(dynamic_cast<const Item*>(t1), dynamic_cast<const Item*>(t2),
ord);
}
template<class Type> bool comparePoint(Type* t1, Type* t2, int ord)
{
	return comparePoint(dynamic_cast<const Trains*>(t1), dynamic_cast<const
Trains*>(t2), ord);
}
template<class Type> bool compareavgTimePeriod(Type* t1, Type* t2, int ord)
{
	return compareavgTimePeriod(dynamic_cast<const Item*>(t1), dynamic_cast<const
Item*>(t2), ord);
}
template<class Type> bool compareStops(Type* t1, Type* t2, int ord)
{
	return compareStops(dynamic_cast<Diesel*>(t1), dynamic_cast<Diesel*>(t2), ord);
}
template<class Type> bool compareVipSeats(Type* t1, Type* t2, int ord)
{
	return compareVipSeats(dynamic_cast<vipTrain*>(t1), dynamic_cast<vipTrain*>(t2),
ord);
}
template<class Type> bool compareTheme(Type* t1, Type* t2, int ord)
```

```
{
	return compareTheme(dynamic_cast<exhTrain*>(t1), dynamic_cast<exhTrain*>(t2), ord);
}
template<class Type> bool compareStorage(Type* t1, Type* t2, int ord)
{
	return compareStorage(dynamic_cast<carryWeight*>(t1),
dynamic_cast<carryWeight*>(t2), ord);
}
template<class Type> bool compareLenght(Type* t1, Type* t2, int ord)
{
	return compareLenght(dynamic_cast<carryWood*>(t1), dynamic_cast<carryWood*>(t2),
ord);
}
template<class Type> bool compareTypeSpeed(Type* t1, Type* t2, int ord)
{
	return compareTypeSpeed(dynamic_cast<SpeedTrain*>(t1),
dynamic_cast<SpeedTrain*>(t2), ord);
}
```

Interface.h

```
#pragma once
#include "List.cpp"

template<class Type>
class Interface
{
	Type obj;			//Шаблонная переменная
public:
	Interface();			//Конструктор без параметров
	Interface(const Type& obj);	//Конструктор копирования
	void mainMenu()                                                    //Главное меню
	{
		char decision;
		int trigger;		//Номер данных
		string fileName;	//Имя файла
		Interface <Diesel> designDiesel;
		Interface <Electro> designElectro;
		Interface <carryWeight> designWeight;
		Interface <carryWood> designWood;
		Interface <exhTrain> designExh;
		Interface <SpeedTrain> designSpeed;


		while (true)
		{
			cout << "С какими данными вы хотите работать?" << endl;
			cout << "1| Обычные поезда" << endl;
			cout << "2| Международные и особые поезда" << endl;
			cout << "3| Промышленные поезда" << endl;
			cout << "4| Выход" << endl;

			cout << "Ваше решение: ";
			cin >> decision;
			switch (decision)
			{
			case '1':
				system("cls");
				cout << "Выберите подкатегорию данных" << endl;
```

```
                        cout << "1| Дизельные поезда" << endl;
                        cout << "2| Электрички" << endl;
                        cout << "3| Назад" << endl;

                        cout << "Ваше решение: ";
                        cin >> decision;
                        switch (decision)
                        {
                        case '1':
                                fileName = "Diesel.txt";
                                trigger = 1;
                                designDiesel.orderMenu(trigger, fileName);      //Вызов
меню
                                break;
                        case '2':
                                fileName = "Electro.txt";
                                trigger = 2;
                                designElectro.orderMenu(trigger, fileName);      //Вызов
меню
                                break;
                        case '3':
                                system("cls");
                                break;
                        default:
                                system("cls");
                                cout << "[ОШИБКА] Такой подкатегории не существует" <<
endl;
                                break;
                        }
                        break;
                case '2':
                        system("cls");
                        cout << "Выберите подкатегорию данных" << endl;
                        cout << "1| Поезда-выставки" << endl;
                        cout << "2| Скоростные поезда" << endl;
                        cout << "3| Назад" << endl;
                        cout << "Ваше решение: ";
                        cin >> decision;
                        switch (decision)
                        {
                        case '1':
                                fileName = "exhTrain.txt";
                                trigger = 3;
                                designExh.orderMenu(trigger, fileName);      //Вызов
меню
                                break;
                        case '2':
                                fileName = "SpeedTrain.txt";
                                trigger = 4;
                                designSpeed.orderMenu(trigger, fileName);      //Вызов
меню
                                break;
                        case '3':
                                system("cls");
                                break;
                        default:
                                system("cls");
                                cout << "[ОШИБКА] Такой подкатегории не существует" <<
endl;
                                break;
```

```
                        }
                        break;
                case '3':
                        system("cls");
                        cout << "Выберите подкатегорию данных" << endl;
                        cout << "1| Грузовые поезда" << endl;
                        cout << "2| Сверхдлинные поезда" << endl;
                        cout << "3| Назад" << endl;
                        cout << "Ваше решение: ";
                        cin >> decision;
                        switch (decision)
                        {
                        case '1':
                                fileName = "carryWeight.txt";
                                trigger = 5;
                                designWeight.orderMenu(trigger, fileName);  //Вызов меню
                                break;
                        case '2':
                                trigger = 6;
                                fileName = "carryWood.txt";
                                designWood.orderMenu(trigger, fileName);//Вызов меню
                                break;
                        case '3':
                                system("cls");
                                break;
                        default:
                                system("cls");
                                cout << "[ОШИБКА] Такой подкатегории не существует" <<
endl;
                                break;
                        }
                        break;
                case '4':
                        exit(0);
                        break;
                default:
                        system("cls");
                        cout << "[ОШИБКА] Такой категории не существует" << endl;
                        break;
                }
        }
}
void orderMenu(int trigger, string fileName)    //Меню для работы с данными
{
        List<Type> list;                        //Объявление списка
        list.readFile(fileName);        //Считывание данных с файла
        Type value;
        char decision;
        int index;
        bool flag = true;
        while (flag)
        {
                cout << "Действия с данными" << endl;
                cout << "1| Добавить элементы" << endl;
                cout << "2| Удалить элементы" << endl;
                cout << "3| Поиск элемента" << endl;
                cout << "4| Просмотреть список" << endl;
                cout << "5| Редактирование списка" << endl;
                cout << "6| Сортировка списка" << endl;
                cout << "7| Сохранить прогресс" << endl;
```

```
                cout << "8| Выход" << endl;
                cout << "Ваше решение: ";
                cin >> decision;
                switch (decision)
                {
                case '1':
                        system("cls");
                        cout << "Выберите действие" << endl;
                        cout << "1| Добавить в начало списка" << endl;
                        cout << "2| Добавить в конец списка" << endl;
                        cout << "Ваше решение: ";
                        cin >> decision;
                        switch (decision)
                        {
                        case '1':
                                system("cls");
                                value.setData();
                                list.push(value);
                                cout << "[ЛОГ] Успешное выполнение действия" << endl;
                                break;
                        case '2':
                                system("cls");
                                value.setData();
                                list.pushBack(value);
                                cout << "[ЛОГ] Успешное выполнение действия" << endl;
                                break;
                        default:
                                cout << "[ОШИБКА] Такой подкатегории не существует" <<
endl;
                                break;
                        }
                        break;
                case '2':
                        system("cls");
                        cout << "Выберите действие" << endl;
                        cout << "1| Удалить с начала списка" << endl;
                        cout << "2| Удалить с конца списка" << endl;
                        cout << "3| Удалить заданный элемент списка" << endl;
                        cout << "Ваше решение: ";
                        cin >> decision;
                        if (list.getCount() == 0)
                        {
                                system("cls");
                                cout << "[ОШИБКА] Список пуст" << endl;
                        }
                        else {
                                switch (decision)
                                {
                                case '1':
                                list.dellOne();
                                        system("cls");
                                        cout << "[ЛОГ] Успешное выполнение действия" <<
endl;
                                        break;
                                case '2':
                                        list.dellOneBack();
                                        system("cls");
                                        cout << "[ЛОГ] Успешное выполнение действия" <<
endl;
                                        break;
```

```
                                        case '3':
                                                system("cls");
                                                value.showHeader();
                                                list.show();
                                                cout << "Введите номер элемента: ";
                                                cin >> index;
                                                list.dellPoint(list[index]);
        //Удаление определенного элемента
                                                system("cls");
                                                cout << "[ЛОГ] Успешное выполнение действия" <<
endl;
                                                break;
                                        default:
                                                system("cls");
                                                cout << "[ОШИБКА] Такой подкатегории не
существует" << endl;
                                                break;
                                        }
                                }
                                break;
                        case '3':
                                system("cls");
                                if (list.getCount() == 0)
                                {
                                        system("cls");
                                        cout << "[ОШИБКА] Список пуст" << endl;
                                }
                                else {
                                        cout << "Введите ключи для поиска" << endl;
                                        value.setOptional();
                                        system("cls");
                                        list.search(value);
                                }
                                break;
                        case '4':
                                system("cls");
                                cout << "Выберите действие" << endl;
                                cout << "1| Просмотреть с начала" << endl;
                                cout << "2| Просмотреть с конца" << endl;
                                cout << "Ваше решение: " << endl;
                                cin >> decision;
                                if (list.getCount() == 0)
                                {
                                        system("cls");
                                        cout << "[ОШИБКА] Список пуст" << endl;
                                }
                                else {
                                        switch (decision)
                                        {
                                        case '1':
                                                system("cls");
                                                value.showHeader();
                                                list.show();
                                                cout << "[ЛОГ] Успешное выполнение действия" <<
endl;
                                                break;
                                        case '2':
                                                system("cls");
                                                value.showHeader();
                                                list.showBack();
```

```
                                        cout << "[ЛОГ] Успешное выполнение действия" <<
endl;
                                        break;
                                default:
                                        system("cls");
                                        cout << "[ОШИБКА] Такой подкатегории не
существует" << endl;
                                        break;
                                }
                        }
                        break;
                case '5':
                        if (list.getCount() == 0)
                        {
                                system("cls");
                                cout << "[ОШИБКА] Список пуст" << endl;
                        }
                        else {
                                system("cls");
                                value.showHeader();
                                list.show();
                                cout << "Введите номер элемента для редактирования: ";
                                cin >> index;
                                cout << "Выберите поле для редактирования" << endl;
                                list[index]->obj.editData();
                                cout << "[ЛОГ] Успешное выполнение действия" << endl;
                        } break;
                case '6':
                        if (list.getCount() == 0)
                        {
                                system("cls");
                                cout << "[ОШИБКА] Список пуст" << endl;
                        }
                        else {
                                cout << "Выберите способ сортировки" << endl;
                                cout << "1| По убыванию" << endl;
                                cout << "2| По возрастанию" << endl;
                                cout << "Ваше решение: ";
                                cin >> decision;
                                switch (decision)
                                {
                                case '1':
                                        system("cls");
                                        cout << "Выберите поле для сортировки" << endl;
                                        value.showLabel();
                                        cout << "Ваше решение: " << endl;
                                        cin >> decision;
                                        switch (decision)
                                        {
                                        case '1':
                                                list.sort(1, compareName<>); break;
                                        case '2':
                                                list.sort(1, compareType<>); break;
                                        case '3':
                                                list.sort(1, comparePrice<>); break;
                                        case '4':
                                                list.sort(1, compareDate<>); break;
                                        case '5':
                                                switch (trigger)
                                                {
```

```
                                        case 1:
                                        case 2:
                                                list.sort(1, compareTime<>); break;
                                        case 3:
                                        case 4:
                                                list.sort(1, comparePoint<>);
break;

                                        } break;
                                case '6':
                                        switch (trigger)
                                        {
                                        case 1:
                                                list.sort(1, compareStops<>);
break;
                                        case 2:
                                                list.sort(1, compareVipSeats<>);
break;
                                        case 3:
                                                list.sort(1, compareTheme<>);
break;
                                        case 4:
                                                list.sort(1, compareStorage<>);
break;
                                        case 5:
                                                list.sort(1, compareLenght<>);
break;
                                        case 6:
                                                list.sort(1, compareTypeSpeed<>);
                                                break;
                                        }
                                        break;
                                default:
                                        system("cls");
                                        cout << "[ОШИБКА] Такой подкатегории не
существует" << endl;
                                        break;
                                }
                                system("cls");
                                cout << "[ЛОГ] Успешное выполнение действия" <<
endl;
                                break;
                        case '2':
                                system("cls");
                                cout << "Выберите поле для сортировки" << endl;
                                value.showLabel();
                                cout << "Ваше решение: " << endl;
                                cin >> decision;
                                switch (decision)
                                {
                                case '1':
                                        list.sort(2, compareName<>); break;
                                case '2':
                                        list.sort(2, compareType<>); break;
                                case '3':
                                        list.sort(2, comparePrice<>); break;
                                case '4':
                                        list.sort(2, compareDate<>); break;
                                case '5':
                                        switch (trigger)
```

```
                                                {
                                                case 1:
                                                case 2:
                                                       list.sort(2, compareTime<>); break;
                                                case 3:
                                                case 4:
                                                       list.sort(2, comparePoint<>);
break;
                                                case '6':
                                                       switch (trigger)
                                                       {
                                                       case 1:
                                                              list.sort(2,
compareStops<>); break;
                                                       case 2:
                                                              list.sort(2,
compareVipSeats<>); break;
                                                       case 3:
                                                              list.sort(2,
compareTheme<>); break;
                                                       case 4:
                                                              list.sort(2,
compareStorage<>); break;
                                                       case 5:
                                                              list.sort(2,
compareLenght<>); break;
                                                       case 6:
                                                              list.sort(2,
compareTypeSpeed<>); break;
                                                       }
                                                       break;
                                                default:
                                                       system("cls");
                                                       cout << "[ОШИБКА] Такой
подкатегории не существует" << endl;
                                                       break;
                                                }
                                                system("cls");
                                                cout << "[ЛОГ] Успешное выполнение
действия" << endl;
                                                break;
                                         default:
                                                system("cls");
                                                cout << "[ОШИБКА] Такой подкатегории не
существует" << endl;
                                         }
                                  }
                                  break;
                           case '7':
                                  system("cls");
                                  cout << "Вы точно желаете сохранить прогресс?" << endl;
                                  cout << "1| Да" << endl;
                                  cout << "2| Нет" << endl;
                                  cout << "Ваше решение: ";
                                  cin >> decision;
                                  switch (decision)
                                  {
                                  case '1':
                                         list.writeFile(fileName);
                                         cout << "[ЛОГ] Успешное сохранение" << endl;
```

```
                                        break;
                                case '2':
                                        break;
                                default:
                                        system("cls");
                                        cout << "[ОШИБКА] Такой категории не существует" <<
endl;
                                        break;
                                }
                                break;
                        case '8': { system("cls");
                                flag = false; }break;
                        default:
                                system("cls");
                                cout << "[ОШИБКА] Такой категории не существует" << endl;
                                break;
                                }
                        }
                }
        }
        ~Interface();                   //Деструктор
};
```

Item.h

```
#include "header.h"

class Item
{
protected:
        string nameItem;                        //Направление
        string seats;                         //Свободные места
        int price;                            //Стоимость билета
        int date;                           //Дата отправления
public:
        Item();                                 //Конструктор без параметров
        Item(const Item&);                      //Конструктор копирования
        Item(string, string, int, int);         //Конструтор с параметрами
        ~Item();                                //Деструктор
        void setData();                         //Метод, определяющий поля элемента
        void showData();                        //Метод, выводящий данные полей элемента
        void editData();                        //Метод, изменяющий данные полей элемента
        void showLabel();                       //Метод, выводящий полей элемента
        virtual string getName();                               //Метод, возвращающий nameItem
        virtual string getType();                               //Метод, возвращающий typeItem
        friend ostream& operator<<(ostream&, Item&);            //Перегрузка вывода
        friend istream& operator >> (istream&, Item&);          //Перегрузка ввода
        friend ofstream& operator<<(ofstream&, Item&);           //Перегрузка вывода
        friend ifstream& operator >> (ifstream&, Item&);         //Перегрузка ввода
        Item operator=(const Item&);                    //Перегрузка оператора присваивания
        bool operator==(const Item&);                  //Перегрузка оператора сравнения
        void showHeader();                //Метод, показывающий заголовки элемента
        friend bool compareName(const Item*, const  Item*, int);
        friend bool compareType(const Item*, const  Item*, int);
        friend bool comparePrice(const Item*, const  Item*, int);
        friend bool compareDate(const Item*, const  Item*, int);
};
```

List.h

```
#include "exhTrain.h"
#include "Electro.h"
#include "Diesel.h"
#include "header.h"
#include "carryWeight.h"
#include "vipTrain.h"
#include "Item.h"
#include "carryWood.h"
#include "Trains.h"
#include "SpeedTrain.h"

template<class Type>
struct Line
{
	Type obj;			//Элемент
	Line<Type>* next;		//Указатель на следующий элемент
	Line<Type>* prev;		//Указатель на предыдущий элемент
};

template<class Type>
class List
{
private:
	Line<Type>* begin;			//Указатель на начало списка
	Line<Type>* end;			//Указатель на конец списка
	int count;				//Количество элементов
public:
	List();					//Конструктор без параметров
	void push(const Type&);			//Добавить элемент в начало
	void pushBack(const Type&);		//Добавить элемент в конец
	Type dellOne();				 //Удалить элемент с начала
	Type dellOneBack();			//Удалить элемент с конца
	void dellAll();				//Удалить все элементы
	int getCount();				//Метод, возвращающий поле count
	Type dellPoint(Line<Type>*);		//Удаление заданного элемента
	void show();				 //Метод, выводящий все элемента списка с начала
	void showBack();			//Метод, выводящий все элемента списка с конца
	Line<Type>* operator [] (int);		//Перегрузка оператора []
	void readFile(string);			//Чтение данных из файла
	void writeFile(string);			//Запись данных в файл
	bool isEmpty();				 //Метод, проверяющий список на пустоту
	void search(const Type&);		//Поиск элемента по названию
	void sort(int, bool(*Compare)(Type*, Type*, int));	//Сортировка по всем полям
	~List();				 //Деструктор
};
```

SpeedTrain.h

```
#include "vipTrain.h"

class SpeedTrain : public vipTrain
{
protected:
	int maxSpeed;			//максимально развиваемая скорость
public:
	SpeedTrain();				//Конструктор без параметров
```

```
	SpeedTrain(const SpeedTrain&);
		//Конструктор копирования
	SpeedTrain(string, string, int, int, string, int);		//Конструктор с
параметрами
	~SpeedTrain();			//Десктрутор
	void setData();			//Метод, определяющий данные поля элемента
	void showLabel();		//Метод, выводящий поля элемента
	void showData();		//Метод, выводящий данные поля элемента
	void setOptional();		//Метод заполнения определенных данных
	friend ostream& operator<<(ostream&, SpeedTrain&);		//Перегрузка
вывода
	friend istream& operator >> (istream&, SpeedTrain&);		//Перегрузка
ввода
	friend ofstream& operator<<(ofstream&, SpeedTrain&);		//Перегрузка
вывода
	friend ifstream& operator >> (ifstream&, SpeedTrain&);		//Перегрузка
ввода
	SpeedTrain operator=(const SpeedTrain&);	//Перегрузка оператора
присваивания
	bool operator==(const SpeedTrain&);	//Перегрузка оператора сравнения
	void showHeader();		//Метод, выводящий заголовки поля элемента
	void editData();		//Метод, изменяющий данные поля элемента
	friend bool compareTypeSpeed(const SpeedTrain*, const SpeedTrain*, int);
//Метод, сравнивающий поле maxSpeed
};
```

Trains.h

```
#include "Item.h"

class Trains : public Item
{
protected:
	int hours;			//Время отправления
public:
	Trains();			//Конструктор без параметров
	Trains(const Trains&);		//Конструктор копирования
	Trains(string, string, int, int, int);	//Конструктор с параметрами
	~Trains();			//Деструктор
	void setData();			//Метод, определяющий поля элемента
	void showData();		//Метод, выводящий данные поля элемента
	void editData();		//Метод, изменяющий данные поля элемента
	void showLabel();		//Метод, выводящий поля элемента
	friend ostream& operator<<(ostream&, Trains&);  //Перегрузка оператора вывода
	friend istream& operator >> (istream&, Trains&); //Перегразка оператора ввода
	Trains operator=(const Trains&);  //Перегрузка оператора присваивания
	bool operator==(const Trains&);	//Перегрузка оператора сравнения
	friend ofstream& operator<<(ofstream&, Trains&);//Перегрузка оператора вывода
	friend ifstream& operator >> (ifstream&, Trains&); //Перегрузка оператора ввода
	void showHeader();		//Метод, выводящий заголовки элемента
	friend bool compareTime(const Trains*, const Trains*, int);	//Метод,
сравнивающий поле hours
};
```

VipTrain.h

```
#include "Item.h"

class vipTrain : public Item
{
protected:
	string vipCount;		//количество мест в VIP вагонах
public:
	vipTrain();			//Конструктор без параметров
	vipTrain(const vipTrain&);	//Конструктор копирования
	vipTrain(string, string, int, int, string);		//Конструктор с
параметрами
	~vipTrain();		//Деструктор
	void setData();		//Метод, определяющий данные поля элемента
	void showData();		//Метод, выводящий данные поля элемента
	void showLabel();		//Метод, выводящий поля элемента
	//void setOptional();		//Метод заполнения определенных данных
	friend ostream& operator<<(ostream&, vipTrain&);		//Перегрузка вывода
	friend istream& operator >> (istream&, vipTrain&);		//Перегрузка ввода
	friend ofstream& operator<<(ofstream&, vipTrain&);		//Перегрузка вывода
	friend ifstream& operator >> (ifstream&, vipTrain&);		//Перегрузка ввода
	vipTrain operator=(const vipTrain&);		//Перегрузка оператора
присваивания
	bool operator==(const vipTrain&);		//Перегрузка оператора сравнения
	friend bool compareVipSeat(const vipTrain*,const  vipTrain*, int);
//Метод, сравнивающий поле vipCount
	void showHeader();		//Метод, выводящий заголовки поля элемента
	void editData();		//Метод, изменяющий данные поля элемента
};
```

Исходные файлы:

CarryWeight.cpp

```
#include "carryWeight.h"

carryWeight::carryWeight() : Item()
{
	amountLoaded = 0;
}
carryWeight::carryWeight(const carryWeight& other) : Item(other)
{
	amountLoaded = other.amountLoaded;
}
carryWeight::carryWeight(string nameItem, string typeItem, int costItem, int countItem,
int amount) : Item(nameItem, typeItem, costItem, countItem)
{
	amountLoaded = amount;
}
void carryWeight::setData()
{
	Item::setData();

	cout << "Введите количество вагонов с грузом: ";
	cin >> this->amountLoaded;
```

```
}
void carryWeight::showHeader()
{
	cout << "|--------------------------------------СПИСОК-----------------------------
---------|" << endl;
	Item::showHeader();
	cout << setw(17) << "Вагонов с грузом" << "|" << endl;
	cout << "|--------------------------------------------------------------------------
---------|" << endl;
}
void carryWeight::showData()
{
	Item::showData();
	cout << setw(17) << left << amountLoaded << "|" << endl;
}
void carryWeight::showLabel()                                          //Вывод полей класса
{
	Item::showLabel();
	cout << "5| Вагонов с грузом" << endl;
}

void carryWeight::editData()
{
	string temp;		//Переменная для проверки на буквы
	char decision;
	Item::editData();	//Метод редактирования данных
	cout << "5| Время в пути" << endl;
	cout << "Ваше решение: ";
	cin >> decision;
	switch (decision)
	{
	case '1':
		cout << "Введите название: ";
		cin >> this->nameItem;
		break;
	case '2':
		cout << "Введите количество мест: ";
		cin >> this->seats;
		break;
	case '3':
		cout << "Введите цену: ";
		cin >> temp;
		if (isFigure(temp))
		{
			this->price = stoi(temp);
		}
		else {
			cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
		}
		break;
	case '4':
		cout << "Введите дату: ";
		cin >> temp;
		if (isFigure(temp))
		{
			this->date = stoi(temp);
		}
		else {
			cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
		}
```

```
			break;
		case '5':
			cout << "Введите количество вагонов с грузом: ";
			cin >> temp;
			if (isFigure(temp))
			{
				this->amountLoaded = stoi(temp);
			}
			else
				cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
			break;
		default:
			cout << "[ОШИБКА] Такого пункта меню не существует" << endl;
			break;
		}
}

void carryWeight::setOptional()
{
		carryWeight value;				//Буферный объект
		char decision = '1';
		while (decision == '1')
		{
			value.editData();		//Метод редактирования данных
			cout << "Желаете продолжить?" << endl;
			cout << "1| Да" << endl;
			cout << "2| Нет" << endl;
			cout << "Ваше решение: ";
			cin >> decision;
			switch (decision)
			{
			case '1':
				break;
			case '2':
				decision = '2';
				break;
			default:
				cout << "[ОШИБКА] Неверное значение" << endl;
				cout << "Желаете продолжить?" << endl;
				cout << "1| Да" << endl;
				cout << "2| Нет" << endl;
				cout << "Ваше решение: ";
				cin >> decision;
				break;
			}
		}
		*this = value;
}

carryWeight carryWeight::operator=(const carryWeight& obj)
{
		this->Item::operator=(obj);
		this->amountLoaded = obj.amountLoaded;
		return *this;
}

bool carryWeight::operator==(const carryWeight& obj)
{
		bool flag = this->Item::operator==(obj);
		if (obj.amountLoaded != NULL && flag != false)
```

```
	{
		if (this->amountLoaded == obj.amountLoaded)
		{
			flag = true;
		}
		else { return false; }
	}
	return flag;
}

ostream& operator<<(ostream& sout, carryWeight& obj)
{
	sout << dynamic_cast<Item&>(obj);                                    //Приведение к
типу
	sout << setw(24) << obj.amountLoaded << "|" << endl;
	return sout;
}

istream& operator >> (istream& sin, carryWeight& obj)
{
	sin >> dynamic_cast<carryWeight&>(obj);      //Приведение к типу
	cout << "Введите количество вагонов с грузом: ";
	sin >> obj.amountLoaded;
	return sin;
}
ofstream& operator<<(ofstream& fout, carryWeight& obj)
{
	fout << dynamic_cast<Item&>(obj);         //Приведение к типу
	fout << obj.amountLoaded << "\n";
	return fout;
}

ifstream& operator >> (ifstream& fin, carryWeight& obj)
{
	fin >> dynamic_cast<Item&>(obj);//Приведение к типу
	fin >> obj.amountLoaded;
	return fin;
}

bool compareTimeStorage(const carryWeight* obj, const carryWeight* obj2, int trigger)
{
	switch (trigger)
			//Выбор как сравнивать
	{
	case 1:
		return obj->amountLoaded < obj2->amountLoaded; //По возрастанию
		break;
	case 2:
		return obj->amountLoaded > obj2->amountLoaded; //По убыванию
		break;
	}
	return false;
}
```

CarryWood.cpp

```
#include "carryWood.h"

carryWood::carryWood() : Item()
{
	amountLenght = 0;
}
carryWood::carryWood(const carryWood& other) : Item(other)
{
	amountLenght = other.amountLenght;
}
carryWood::carryWood(string nameItem, string typeItem, int costItem, int countItem, int
amount) : Item(nameItem, typeItem, costItem, countItem)
{
	amountLenght = amount;
}
void carryWood::setData()
{
	Item::setData();
	cout << "Введите количество вагонов с грузом: ";
	cin >> this->amountLenght;
}
void carryWood::showHeader()
{
	cout << "|--------------------------------------СПИСОК-----------------------------
---------|" << endl;
	Item::showHeader();
	cout << setw(14) << "Длина вагонов" << "|" << endl;
	cout << "|--------------------------------------------------------------------------
---------|" << endl;
}
void carryWood::showData()
{
	Item::showData();
	cout << setw(14) << left << amountLenght << "|" << endl;
}
void carryWood::showLabel()                                    //Вывод полей класса
{
	Item::showLabel();
	cout << "5| Длина вагонов" << endl;
}

void carryWood::editData()
{
	string temp;		//Переменная для проверки на буквы
	char decision;
	Item::editData();	//Метод редактирования данных
	cout << "5| Длина вагонов" << endl;
	cout << "Ваше решение: ";
	cin >> decision;
	switch (decision)
	{
	case '1':
		cout << "Введите название: ";
		cin >> this->nameItem;
		break;
	case '2':
		cout << "Введите количество мест: ";
```

```
			cin >> this->seats;
			break;
		case '3':
			cout << "Введите цену: ";
			cin >> temp;
			if (isFigure(temp))
			{
				this->price = stoi(temp);
			}
			else {
				cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
			}
			break;
		case '4':
			cout << "Введите дату: ";
			cin >> temp;
			if (isFigure(temp))
			{
				this->date = stoi(temp);
			}
			else {
				cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
			}
			break;
		case '5':
			cout << "Введите общую длину вагонов: ";
			cin >> temp;
			if (isFigure(temp))
			{
				this->amountLenght = stoi(temp);
			}
			else
				cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
			break;
		default:
			cout << "[ОШИБКА] Такого пункта меню не существует" << endl;
			break;
		}
}

void carryWood::setOptional()
{
		carryWood value;				//Буферный объект
		char decision = '1';
		while (decision == '1')
		{
			value.editData();		//Метод редактирования данных
			cout << "Желаете продолжить?" << endl;
			cout << "1| Да" << endl;
			cout << "2| Нет" << endl;
			cout << "Ваше решение: ";
			cin >> decision;
			switch (decision)
			{
			case '1':
				break;
			case '2':
				decision = '2';
				break;
			default:
```

```
                        cout << "[ОШИБКА] Неверное значение" << endl;
                        cout << "Желаете продолжить?" << endl;
                        cout << "1| Да" << endl;
                        cout << "2| Нет" << endl;
                        cout << "Ваше решение: ";
                        cin >> decision;
                        break;
                }
        }
        *this = value;
}

carryWood carryWood::operator=(const carryWood& obj)
{
        this->Item::operator=(obj);
        this->amountLenght = obj.amountLenght;
        return *this;
}

bool carryWood::operator==(const carryWood& obj)
{
        bool flag = this->Item::operator==(obj);
        if (obj.amountLenght != NULL && flag != false)
        {
                if (this->amountLenght == obj.amountLenght)
                {
                        flag = true;
                }
                else { return false; }
        }
        return flag;
}

ostream& operator<<(ostream& sout, carryWood& obj)
{
        sout << dynamic_cast<Item&>(obj);                                       //Приведение к
типу
        sout << setw(14) << obj.amountLenght << "|" << endl;
        return sout;
}

istream& operator >> (istream& sin, carryWood& obj)
{
        sin >> dynamic_cast<carryWood&>(obj);        //Приведение к типу
        cout << "Введите общую длину вагонов: ";
        sin >> obj.amountLenght;
        return sin;
}
ofstream& operator<<(ofstream& fout, carryWood& obj)
{
        fout << dynamic_cast<Item&>(obj);         //Приведение к типу
        fout << obj.amountLenght << "\n";
        return fout;
}

ifstream& operator >> (ifstream& fin, carryWood& obj)
{
        fin >> dynamic_cast<Item&>(obj);//Приведение к типу
        fin >> obj.amountLenght;
        return fin;
```

```
}

bool compareLong(const carryWood* obj, const carryWood* obj2, int trigger)
{
	switch (trigger)
			//Выбор как сравнивать
	{
	case 1:
		return obj->amountLenght < obj2->amountLenght; //По возрастанию
		break;
	case 2:
		return obj->amountLenght > obj2->amountLenght; //По убыванию
		break;
	}
	return false;
}
```

Diesel.cpp

```
#include "Diesel.h"

Diesel::Diesel()
{
	addStops = "";
}
Diesel::Diesel(const Diesel& other)
{
	addStops = other.addStops;
}
Diesel::Diesel(string nameItem, string typeItem, int costItem, int countItem, int hours,
string add) : Trains::Trains(nameItem, typeItem, costItem, countItem, hours)
{
	this->addStops = add;
}
Diesel::~Diesel() = default;
void Diesel::setData()
{
	string temp;
	Trains::setData();
	cout << "Введите дополнительные остановки: " << endl;
	cin >> temp;
	this->addStops = temp;
}

void Diesel::showHeader()
{
	cout << "|--------------------------------------СПИСОК---------------------------
---------|" << endl;
	Trains::showHeader();
	cout << setw(35) << "Дополнительные остановки" << "|" << endl;
	cout << "|--------------------------------------------------------------------------
---------|" << endl;
}
void Diesel::showData()
{
	Trains::showData();
	cout << setw(35) << addStops << "|";
}
```

```
void Diesel::editData()
{
	Trains::editData();
	cout << "6| Дополнительные остановки" << endl;
}
void Diesel::showLabel()
{
	Trains::showLabel();
	cout << "6| Дополнительные остановки" << endl;

}

Diesel Diesel::operator=(const Diesel& obj)
{
	this->nameItem = obj.nameItem;
	this->seats = obj.seats;
	this->price = obj.price;
	this->date = obj.date;
	this->hours = obj.hours;
	this->addStops = obj.addStops;
	return *this;
}
bool Diesel::operator==(const Diesel& obj)
{
	bool flag = true;
	if (obj.nameItem != "")
	{
		if (this->nameItem == obj.nameItem)
		{
			flag = true;
		}
		else { return false; }
	}
	if (obj.seats != "")
	{
		if (this->seats == obj.seats)
		{
			flag = true;
		}
		else { return false; }
	}
	if (obj.price != NULL)
	{
		if (this->price == obj.price)
		{
			flag = true;
		}
		else { return false; }
	}
	if (obj.date != NULL)
	{
		if (this->date == obj.date)
		{
			flag = true;
		}
		else { return false; }
	}
	if (obj.hours != NULL)
	{
		if (this->hours == obj.hours)
```

```
		{
			flag = true;
		}
		else { return false; }
	}
	if (obj.addStops != "")
	{
		if (this->addStops == obj.addStops)
		{
			flag = true;
		}
		else { return false; }
	}
	return flag;
}

ostream& operator<<(ostream& sout, Diesel& obj)
{
	sout << "|" << setw(21) << left << obj.nameItem << "|" << setw(16) << obj.seats <<
"|" << setw(6) << obj.price << "|" << setw(7) << obj.date << "|" << setw(18) << obj.hours
<< "|" << setw(35) << obj.addStops << "|";
	return sout;
}
istream& operator >> (istream& sin, Diesel& obj)
{
	string temp;
	cout << "Введите направление: ";
	sin >> obj.nameItem;
	cout << "Введите к-во мест: ";
	sin >> obj.seats;
	cout << "Введите цену: ";
	sin >> temp;
	while (!isFigure(temp))
	{
		cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
		cout << "Введите год: ";
		cin >> temp;
	}
	obj.price = stoi(temp);
	cout << "Введите дату: ";
	sin >> temp;
	while (!isFigure(temp))
	{
		cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
		cout << "Введите дату: ";
		cin >> temp;
	}
	obj.date = stoi(temp);

	cout << "Введите максимальную скорость: ";
	sin >> temp;
	while (!isFigure(temp))
	{
		cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
		cout << "Введите максимальную скорость: ";
		cin >> temp;
	}
	obj.hours = stoi(temp);
	cout << "Введите дополнительные остановки" << endl;
	sin >> obj.addStops;
```

```
	return sin;
}
ofstream& operator<<(ofstream& fout, Diesel& obj)
{
	fout << obj.nameItem << " " << obj.seats << " " << obj.price << " " << obj.date << "
" << obj.hours << " " << obj.addStops;
	return fout;
}
ifstream& operator >> (ifstream& fin, Diesel& obj)
{
	fin >> obj.nameItem;
	fin >> obj.seats;
	fin >> obj.price;
	fin >> obj.date;
	fin >> obj.hours;
	fin >> obj.addStops;
	return fin;
}

bool compareStops(const Diesel* obj, const Diesel* obj2, int trigger)
{
	switch (trigger)
	{
	case 1:
		return obj->addStops.length() < obj2->addStops.length();
		break;
	case 2:
		return obj->addStops.length() > obj2->addStops.length();
		break;
	}
	return false;
}

void Diesel::setOptional()
{
	Diesel value;		//Буферный объект
	char decision = '1';
	while (decision == '1')
	{
		value.editData();	//Метод редактирования данных
		cout << "Желаете продолжить?" << endl;
		cout << "1| Да" << endl;
		cout << "2| Нет" << endl;
		cout << "Ваше решение: ";
		cin >> decision;
		switch (decision)
		{
		case '1':
			break;
		case '2':
			decision = '2';
			break;
		default:
			cout << "[ОШИБКА] Неверное значение" << endl;
			cout << "Желаете продолжить?" << endl;
			cout << "1| Да" << endl;
			cout << "2| Нет" << endl;
			cout << "Ваше решение: ";
			cin >> decision;
			break;
```

```
            }
        }
        *this = value;
}
```

Electro.cpp

```
#include "Electro.h"
#include <iostream>
using namespace std;
Electro::Electro() : Trains()
{
        waypointTime = 0;
}

Electro::Electro(const Electro& obj) : Trains(obj)
{
        this->waypointTime = obj.waypointTime;
}

Electro::Electro(string nameItem, string typeItem, int costItem, int countItem, int
lifeTime, int wpTime) : Trains(nameItem, typeItem, costItem, countItem, lifeTime)
{
        this->waypointTime = wpTime;
}

Electro::~Electro() = default;
void Electro::setData()
{
        Trains::setData();
        cout << "Введите время между остановками: ";
        cin >> this->waypointTime;
}

void Electro::showHeader()
{
        cout << "|--------------------------------------СПИСОК---------------------------
---------|" << endl;
        Trains::showHeader();
        cout << setw(24) << "Время между остановками" << "|" << endl;
        cout << "|-------------------------------------------------------------------------
---------|" << endl;
}

void Electro::showData()
{
        Trains::showData();
        cout << setw(24) << left << waypointTime << "|" << endl;
}

void Electro::setOptional()
{
        Electro value;                    //Буферный объект
        char decision = '1';
        while (decision == '1')
        {
                value.editData();        //Метод редактирования данных
                cout << "Желаете продолжить?" << endl;
                cout << "1| Да" << endl;
                cout << "2| Нет" << endl;
```

```
                cout << "Ваше решение: ";
                cin >> decision;
                switch (decision)
                {
                case '1':
                        break;
                case '2':
                        decision = '2';
                        break;
                default:
                        cout << "[ОШИБКА] Неверное значение" << endl;
                        cout << "Желаете продолжить?" << endl;
                        cout << "1| Да" << endl;
                        cout << "2| Нет" << endl;
                        cout << "Ваше решение: ";
                        cin >> decision;
                        break;
                }
        }
        *this = value;
}

void Electro::editData()
{
        string temp;         //Переменная для проверки на буквы
        char decision;
        Trains::editData();  //Метод редактирования данных
        cout << "6| Время в пути" << endl;
        cout << "Ваше решение: ";
        cin >> decision;
        switch (decision)
        {
        case '1':
                cout << "Введите название: ";
                cin >> this->nameItem;
                break;
        case '2':
                cout << "Введите количество мест: ";
                cin >> this->seats;
                break;
        case '3':
                cout << "Введите цену: ";
                cin >> temp;
                if (isFigure(temp))
                {
                        this->price = stoi(temp);
                }
                else {
                        cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
                }
                break;
        case '4':
                cout << "Введите дату: ";
                cin >> temp;
                if (isFigure(temp))
                {
                        this->date = stoi(temp);
                }
                else {
                        cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
```

```
			}
			break;
		case '5':
			cout << "Введите время в пути: ";
			cin >> hours;
			if (isFigure(temp))
			{
				this->hours = stoi(temp);
			}
			else {

				cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;

			}
			break;
		case '6':
			cout << "Введите время между остановками: ";
			cin >> temp;
			if (isFigure(temp))
			{
				this->waypointTime = stoi(temp);
			}
			else {
				cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
			}
			break;
		default:
			cout << "[ОШИБКА] Такого пункта меню не существует" << endl;
			break;
		}
}

void Electro::showLabel()                                    //Вывод полей класса
{
	Trains::showLabel();
	cout << "6| Время в пути" << endl;
}

Electro Electro::operator=(const Electro& obj)
{
	this->Trains::operator=(obj);
	this->waypointTime = obj.waypointTime;
	return *this;
}

bool Electro::operator==(const Electro& obj)
{
	bool flag = this->Trains::operator==(obj);
	if (obj.waypointTime != NULL && flag != false)
	{
		if (this->waypointTime == obj.waypointTime)
		{
			flag = true;
		}
		else { return false; }
	}
	return flag;
}

ostream& operator<<(ostream& sout, Electro& obj)
```

```
{
	sout << dynamic_cast<Trains&>(obj);
	//Приведение к типу
	sout << setw(24) << obj.waypointTime << "|" << endl;
	return sout;
}

istream& operator >> (istream& sin, Electro& obj)
{
	sin >> dynamic_cast<Trains&>(obj);    //Приведение к типу
	cout << "Введите время между остановками: ";
	sin >> obj.waypointTime;
	return sin;
}
ofstream& operator<<(ofstream& fout, Electro& obj)
{
	fout << dynamic_cast<Trains&>(obj);              //Приведение к типу
	fout << obj.waypointTime << "\n";
	return fout;
}

ifstream& operator >> (ifstream& fin, Electro& obj)
{
	fin >> dynamic_cast<Trains&>(obj);//Приведение к типу
	fin >> obj.waypointTime;
	return fin;
}

bool compareWpTime(const Electro* obj, const Electro* obj2, int trigger)
{
	switch (trigger)
			//Выбор как сравнивать
	{
	case 1:
		return obj->waypointTime < obj2->waypointTime; //По возрастанию
		break;
	case 2:
		return obj->waypointTime > obj2->waypointTime; //По убыванию
		break;
	}
	return false;
}
```

exhTrain.cpp

```
#include "exhTrain.h"

exhTrain::exhTrain() : vipTrain::vipTrain()
{
	descExh = "";
}

exhTrain::exhTrain(const exhTrain& obj) : vipTrain(obj)
{
	this->descExh = obj.descExh;
}

exhTrain::exhTrain(string nameItem, string typeItem, int costItem, int countItem, string
storageCondition, string typeMetal) : vipTrain(nameItem, typeItem, costItem, countItem,
storageCondition)
```

```
{
	this->descExh = typeMetal;
}

exhTrain::~exhTrain() {}
void exhTrain::setData()
{
	vipTrain::setData();
	cout << "Введите описание выставки: ";
	cin >> this->descExh;
}

void exhTrain::showLabel()
{
	vipTrain::showLabel();
	cout << "6| Jписание выставки" << endl;
}

void exhTrain::showHeader()
{
	cout << "|-----------------------------------------СПИСОК-------------------------
-------------|" << endl;
	vipTrain::showHeader();
	cout << setw(15) << "Основные события" << "|" << endl;
	cout << "|-----------------------------------------------------------------------
-------------|" << endl;
}

void exhTrain::showData()
{
	vipTrain::showData();
	cout << setw(15) << left << descExh << "|" << endl;
}

exhTrain exhTrain::operator=(const exhTrain& obj)
{
	vipTrain::operator=(obj);
	this->descExh = obj.descExh;
	return *this;
}

bool exhTrain::operator==(const exhTrain& obj)
{
	bool flag = this->vipTrain::operator==(obj);
	if (obj.descExh != "" && flag != false)
	{
		if (this->descExh == obj.descExh)
		{
			flag = true;
		}
		else { return false; }
	}
	return flag;
}

ostream& operator<<(ostream& sout, exhTrain& obj)
{
	sout << dynamic_cast<vipTrain&>(obj);
	sout << setw(15) << obj.descExh << "|" << endl;
	return sout;
```

```
}

istream& operator >> (istream& sin, exhTrain& obj)
{
	sin >> dynamic_cast<vipTrain&>(obj);
	cout << "Введите описание выставки: ";
	sin >> obj.descExh;
	return sin;
}
ofstream& operator<<(ofstream& fout, exhTrain& obj)
{
	fout << dynamic_cast<vipTrain&>(obj);
	fout << obj.descExh << "\n";
	return fout;
}

ifstream& operator >> (ifstream& fin, exhTrain& obj)
{
	fin >> dynamic_cast<vipTrain&>(obj);
	fin >> obj.descExh;
	return fin;
}

bool comparedescExh(const exhTrain* obj, const exhTrain* obj2, int trigger)
{
	switch (trigger)
	{
	case 1:
		return obj->descExh.length() < obj2->descExh.length();
		break;
	case 2:
		return obj->descExh.length() > obj2->descExh.length();
		break;
	}
	return false;
}
void exhTrain::editData()
{
	string temp;
	char decision;
	vipTrain::editData();
	cout << "6| Описание выставки" << endl;
	cout << "Ваше решение: ";
	cin >> decision;
	switch (decision)
	{
	case '1':
		cout << "Введите направление: ";
		cin >> this->nameItem;
		break;
	case '2':
		cout << "Введите количество мест: ";
		cin >> this->seats;
		break;
	case '3':
		cout << "Введите цену: ";
		cin >> temp;
		if (isFigure(temp))
		{
			this->price = stoi(temp);
```

```
                }
                else {
                        cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
                }
                break;
        case '4':
                cout << "Введите дату: ";
                cin >> temp;
                if (isFigure(temp))
                {
                        this->date = stoi(temp);
                }
                else {
                        cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
                }
                break;
        case '5':
                cout << "Введите доп места: ";
                cin >> this->vipCount;
                break;
        case '6':
                cout << "Введите описание выставки: ";
                cin >> this->descExh;
                break;
        default:
                cout << "[ОШИБКА] Такого пункта меню не существует" << endl;
                break;
        }
}
void exhTrain::setOptional()
{
        exhTrain value;
        char decision = '1';
        while (decision == '1')
        {
                value.editData();
                cout << "Желаете продолжить?" << endl;
                cout << "1| Да" << endl;
                cout << "2| Нет" << endl;
                cout << "Ваше решение: ";
                cin >> decision;
                switch (decision)
                {
                case '1':
                        break;
                case '2':
                        decision = '2';
                        break;
                default:

                        cout << "[ОШИБКА] Неверное значение" << endl;
                        cout << "Желаете продолжить?" << endl;
                        cout << "1| Да" << endl;
                        cout << "2| Нет" << endl;
                        cout << "Ваше решение: ";
                        cin >> decision;
                        break;
                }
        }
        *this = value;}
```

Header.cpp

```
#include "exhTrain.h"

exhTrain::exhTrain() : vipTrain::vipTrain()
{
	descExh = "";
}

exhTrain::exhTrain(const exhTrain& obj) : vipTrain(obj)
{
	this->descExh = obj.descExh;
}

exhTrain::exhTrain(string nameItem, string typeItem, int costItem, int countItem, string
storageCondition, string typeMetal) : vipTrain(nameItem, typeItem, costItem, countItem,
storageCondition)
{
	this->descExh = typeMetal;
}

exhTrain::~exhTrain() {}
void exhTrain::setData()
{
	vipTrain::setData();
	cout << "Введите описание выставки: ";
	cin >> this->descExh;
}

void exhTrain::showLabel()
{
	vipTrain::showLabel();
	cout << "6| Jписание выставки" << endl;
}

void exhTrain::showHeader()
{
	cout << "|-----------------------------------------СПИСОК-------------------------
-------------|" << endl;
	vipTrain::showHeader();
	cout << setw(15) << "Основные события" << "|" << endl;
	cout << "|--------------------------------------------------------------------------
-------------|" << endl;
}

void exhTrain::showData()
{
	vipTrain::showData();
	cout << setw(15) << left << descExh << "|" << endl;
}

exhTrain exhTrain::operator=(const exhTrain& obj)
{
	vipTrain::operator=(obj);
	this->descExh = obj.descExh;
	return *this;
}

bool exhTrain::operator==(const exhTrain& obj)
{
```

```
	bool flag = this->vipTrain::operator==(obj);
	if (obj.descExh != "" && flag != false)
	{
		if (this->descExh == obj.descExh)
		{
			flag = true;
		}
		else { return false; }
	}
	return flag;
}

ostream& operator<<(ostream& sout, exhTrain& obj)
{
	sout << dynamic_cast<vipTrain&>(obj);
	sout << setw(15) << obj.descExh << "|" << endl;
	return sout;
}

istream& operator >> (istream& sin, exhTrain& obj)
{
	sin >> dynamic_cast<vipTrain&>(obj);
	cout << "Введите описание выставки: ";
	sin >> obj.descExh;
	return sin;
}
ofstream& operator<<(ofstream& fout, exhTrain& obj)
{
	fout << dynamic_cast<vipTrain&>(obj);
	fout << obj.descExh << "\n";
	return fout;
}

ifstream& operator >> (ifstream& fin, exhTrain& obj)
{
	fin >> dynamic_cast<vipTrain&>(obj);
	fin >> obj.descExh;
	return fin;
}

bool comparedescExh(const exhTrain* obj, const exhTrain* obj2, int trigger)
{
	switch (trigger)
	{
	case 1:
		return obj->descExh.length() < obj2->descExh.length();
		break;
	case 2:
		return obj->descExh.length() > obj2->descExh.length();
		break;
	}
	return false;
}
void exhTrain::editData()
{
	string temp;
	char decision;
	vipTrain::editData();
	cout << "6| Описание выставки" << endl;
	cout << "Ваше решение: ";
```

```
        cin >> decision;
        switch (decision)
        {
        case '1':
               cout << "Введите направление: ";
               cin >> this->nameItem;
               break;
        case '2':
               cout << "Введите количество мест: ";
               cin >> this->seats;
               break;
        case '3':
               cout << "Введите цену: ";
               cin >> temp;
               if (isFigure(temp))
               {
                      this->price = stoi(temp);
               }
               else {
                      cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
               }
               break;
        case '4':
               cout << "Введите дату: ";
               cin >> temp;
               if (isFigure(temp))
               {
                      this->date = stoi(temp);
               }
               else {
                      cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
               }
               break;
        case '5':
               cout << "Введите доп места: ";
               cin >> this->vipCount;
               break;
        case '6':
               cout << "Введите описание выставки: ";
               cin >> this->descExh;
               break;
        default:
               cout << "[ОШИБКА] Такого пункта меню не существует" << endl;
               break;
        }
}
void exhTrain::setOptional()
{
        exhTrain value;
        char decision = '1';
        while (decision == '1')
        {
               value.editData();
               cout << "Желаете продолжить?" << endl;
               cout << "1| Да" << endl;
               cout << "2| Нет" << endl;
               cout << "Ваше решение: ";
               cin >> decision;
               switch (decision)
               {
```

```
                case '1':
                        break;
                case '2':
                        decision = '2';
                        break;
                default:

                        cout << "[ОШИБКА] Неверное значение" << endl;
                        cout << "Желаете продолжить?" << endl;
                        cout << "1| Да" << endl;
                        cout << "2| Нет" << endl;
                        cout << "Ваше решение: ";
                        cin >> decision;
                        break;
                }
        }
        *this = value;
}
```

Interface.cpp

```
#include "Interface.h"

template<class Type>
Interface<Type>::Interface() { }

template<class Type>
Interface<Type>::Interface(const Type& obj)
{
        this->obj = obj.obj;
}

template<class Type>
Interface<Type>::~Interface() = default;
```

Item.cpp

```
#include "Interface.h"

template<class Type>
Interface<Type>::Interface() { }

template<class Type>
Interface<Type>::Interface(const Type& obj)
{
        this->obj = obj.obj;
}

template<class Type>
Interface<Type>::~Interface() = default;
```

List.cpp

```
#include "List.h"

template<class Type>
List<Type>::List()
{
        count = 0;
        begin = NULL;
        end = NULL;
```

```
}

template<class Type>
List<Type>::~List()
{
	Line<Type>* tmp = nullptr;
	if (begin != NULL)
	{
		while (begin != NULL)
		{
			tmp = begin->next;
			delete begin;
			begin = tmp;
		}
	}
}

template<class Type>
void List<Type>::push(const Type& obj)
{
	Line<Type>* temp = new Line<Type>; //Выделение памяти для объекта
	temp->next = NULL;
	Line<Type>* current = nullptr;
	temp->obj = obj;
	count++;
	if (begin != NULL)
	{
		temp->prev = end;
		end->next = temp;
		end = temp;
		current = end;
	}
	else
	{
		temp->prev = NULL;
		begin = end = temp;
	}
}
template<class Type>
void List<Type>::pushBack(const Type& obj)
{
	Line<Type>* current = nullptr;
	Line<Type>* temp = new Line<Type>;
	temp->prev = NULL;
	current = NULL;
	temp->obj = obj;
	count++;
	if (begin != NULL)
	{
		begin->prev = temp;
		temp->next = begin;
		begin = temp;
		current = begin;
	}
	else
	{
		temp->next = NULL;
		begin = end = temp;
	}
}
```

```
template<class Type>
void List<Type>::show()
{
	Line<Type>* current = nullptr;
	current = end;
	cout << current->obj << endl;
	while (current->prev != NULL)
	{
		current = current->prev;
		cout << current->obj << endl;
	}
}
template<class Type>
void List<Type>::showBack()
{
	Line<Type>* current = nullptr;
	current = begin;
	cout << current->obj << endl;
	while (current->next != NULL)
	{
		current = current->next;
		cout << current->obj << endl;
	}
}
template<class Type>
Type List<Type>::dellPoint(Line<Type>* obj)
{
	count--;
	Line<Type>* temp, * tempn, data;
	if (obj == begin && obj != end) //Если удаляемый объект является началом списка
	{
		tempn = obj->next;
		tempn->prev = NULL;
		begin = tempn;
		data.obj = obj->obj;
		delete obj;
		return data.obj;
	}
	else if (obj == end && obj != begin)//Если удаляемый объект является концом списка
	{
		temp = obj->prev;
		temp->next = obj->next;
		end = temp;
		data.obj = obj->obj;
		delete obj;
		return data.obj;
	}
	else if (obj == end && obj == begin)   //Если объект является и концом и началом
	{
		temp = obj;
		data.obj = obj->obj;
		delete end;
		end = begin = NULL;
		return data.obj;
	}
	else
	{
		temp = obj->prev;
		tempn = obj->next;
		temp->next = tempn;
```

```
                tempn->prev = temp;
                data.obj = obj->obj;
                delete obj;
                return data.obj;
        }
}
template<class Type>
Type List<Type>::dellOne()
{
        if (end == begin)
        {
                count--;
                Line<Type> data;
                data.obj = end->obj;
                delete end;
                end = begin = NULL;
                return data.obj;
        }
        else {
                Line<Type>* current = nullptr;
                count--;
                Line<Type> data;
                current = end->prev;
                current->next = NULL;
                data.obj = end->obj;
                delete end;
                end = current;
                return data.obj;
        }
}
template<class Type>
Type List<Type>::dellOneBack()
{
        if (end == begin)
        {
                count--;
                Line<Type> data;
                data.obj = end->obj;
                delete end;
                end = begin = NULL;
                return data.obj;
        }
        else {
                Line<Type>* current = nullptr;
                Line<Type> data;
                count--;
                current = begin->next;
                current->prev = NULL;
                data.obj = end->obj;
                delete begin;
                begin = current;
                return data.obj;
        }
}
template<class Type>
bool List<Type>::isEmpty()
{
        if (begin != NULL)
        {
                return true;
```

```
	}
	else { return false; }
}
template<class Type>
void List<Type>::search(const Type& obj)
{
	int index = 0;
	Line<Type>* current = end;
	while (current != NULL)
	{
		if (current->obj == obj)
		{
			index++;
			if (index == 1)
			{
				current->obj.showHeader();
			}
			cout << current->obj;
		}
		current = current->prev;
	}
	if (index == 0)
	{
		cout << "[ОШИБКА] Элементы с таким именем не существуют" << endl;
	}
}
template<class Type>
Line<Type>* List<Type>::operator[] (int id)
{
	Line<Type>* current = nullptr;
	current = end;
	int index = 1;
	while (id != index)
	{
		if (current->prev != NULL)
		{
			current = current->prev;
		}
		index++;
	}
	return current;
}
template<class Type>
void List<Type>::readFile(string fileName)
{
	ifstream fin(fileName, ios::in);
	Type temp;
	while (fin >> temp)
	{
		this->push(temp);
	}
	fin.close();
}
template<class Type>
void List<Type>::writeFile(string fileName)
{
	ofstream fout(fileName, ios_base::trunc);
	Line<Type>* current = begin;
	while (current != NULL)
	{
```

```
			fout << current->obj;
			current = current->next;
		}
		fout.close();
}
template<class Type>
void List<Type>::sort(int trigger, bool(*Compare)(Type*, Type*, int))
{
		Line<Type>* current = end;
		Line<Type>* temp = nullptr;
		for (current; current != NULL; current = current->prev)
		{
			for (temp = end; temp != NULL; temp = temp->prev)
			{
				if ((*Compare)(&current->obj, &temp->obj, trigger))
				{
					swap(current->obj, temp->obj);
				}
			}
		}
}
template<class Type>
void List<Type>::dellAll()
{
		Line<Type>* tmp = nullptr;
		if (begin != NULL)
		{
			while (begin != NULL)
			{
				tmp = begin->next;
				delete begin;
				begin = tmp;
			}
		}
}
template<class Type>
int List<Type>::getCount()
{
		return this->count;
}
```

SpeedTrain.cpp

```
#include "SpeedTrain.h"

SpeedTrain::SpeedTrain() : vipTrain::vipTrain()
{
		maxSpeed = 0;
}

SpeedTrain::SpeedTrain(const SpeedTrain& obj) : vipTrain(obj)
{
		this->maxSpeed = obj.maxSpeed;
}

SpeedTrain::SpeedTrain(string nameItem, string typeItem, int costItem, int countItem,
string storageCondition, int maxSpeed) : vipTrain(nameItem, typeItem, costItem, countItem,
storageCondition)
{
		this->maxSpeed = maxSpeed;
```

```
}

SpeedTrain::~SpeedTrain() {}
void SpeedTrain::setData()
{
	vipTrain::setData();
	cout << "Введите максимальную скорость: ";
	cin >> this->maxSpeed;
}

void SpeedTrain::showLabel()
{
	vipTrain::showLabel();
	cout << "6| Максимальная скорость" << endl;
}

void SpeedTrain::showHeader()
{
	cout << "|----------------------------------------СПИСОК-------------------------
-------------|" << endl;
	vipTrain::showHeader();
	cout << setw(22) << "Максимальная скорость" << "|" << endl;
	cout << "|--------------------------------------------------------------------------
-------------|" << endl;
}

void SpeedTrain::showData()
{
	vipTrain::showData();
	cout << setw(22) << left << maxSpeed << "|" << endl;
}

SpeedTrain SpeedTrain::operator=(const SpeedTrain& obj)
{
	vipTrain::operator=(obj);
	this->maxSpeed = obj.maxSpeed;
	return *this;
}

bool SpeedTrain::operator==(const SpeedTrain& obj)
{
	bool flag = this->vipTrain::operator==(obj);
	if (obj.maxSpeed != 0 && flag != false)
	{
		if (this->maxSpeed == obj.maxSpeed)
		{
			flag = true;
		}
		else { return false; }
	}
	return flag;
}

ostream& operator<<(ostream& sout, SpeedTrain& obj)
{
	sout << dynamic_cast<vipTrain&>(obj);
	sout << setw(22) << obj.maxSpeed << "|" << endl;
	return sout;
}
```

```
istream& operator >> (istream& sin, SpeedTrain& obj)
{
	string temp;
	sin >> dynamic_cast<vipTrain&>(obj);
	cout << "Введите максимальную скорость: ";
	sin >> temp;
	while (isFigure(temp))
	{
		cout << "[ОШИБКА] Введены неверные символы!" << endl;
		cout << "Введите максимальную скорость: ";
		cin >> temp;
	}
	obj.maxSpeed = stoi(temp);
	return sin;
}
ofstream& operator<<(ofstream& fout, SpeedTrain& obj)
{
	fout << dynamic_cast<SpeedTrain&>(obj);
	fout << obj.maxSpeed << "\n";
	return fout;
}

ifstream& operator >> (ifstream& fin, SpeedTrain& obj)
{
	fin >> dynamic_cast<SpeedTrain&>(obj);
	fin >> obj.maxSpeed;
	return fin;
}

bool compareTypeSpeed(const SpeedTrain* obj, const SpeedTrain* obj2, int trigger)
{
	switch (trigger)
	{
	case 1:
		return obj->maxSpeed < obj2->maxSpeed;
		break;
	case 2:
		return obj->maxSpeed > obj2->maxSpeed;
		break;
	}
	return false;
}
void SpeedTrain::editData()
{
	string temp;
	char decision;
	vipTrain::editData();
	cout << "6| Максимальная скорость" << endl;
	cout << "Ваше решение: ";
	cin >> decision;
	switch (decision)
	{
	case '1':
		cout << "Введите направление: ";
		cin >> this->nameItem;
		break;
	case '2':
		cout << "Введите количество мест: ";
		cin >> this->seats;
		break;
```

```
	case '3':
		cout << "Введите цену: ";
		cin >> temp;
		if (isFigure(temp))
		{
			this->price = stoi(temp);
		}
		else {
			cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
		}
		break;
	case '4':
		cout << "Введите дату: ";
		cin >> temp;
		if (isFigure(temp))
		{
			this->date = stoi(temp);
		}
		else {
			cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
		}
		break;
	case '5':
		cout << "Введите доп места: ";
		cin >> this->vipCount;
		break;
	case '6':
		cout << "Введите максимальную скорость: ";
		cin >> temp;
		if (isFigure(temp))
		{
			this->maxSpeed = stoi(temp);
		}
		else
		{
			cout << "[ОШИБКА] Присутствуют запрещенные символы" << endl;
		}
		break;
	default:
		cout << "[ОШИБКА] Такого пункта меню не существует" << endl;
		break;
	}
}
void SpeedTrain::setOptional()
{
	SpeedTrain value;
	char decision = '1';
	while (decision == '1')
	{
		value.editData();
		cout << "Желаете продолжить?" << endl;
		cout << "1| Да" << endl;
		cout << "2| Нет" << endl;
		cout << "Ваше решение: ";
		cin >> decision;
		switch (decision)
		{
		case '1':
			break;
		case '2':
```

```
                        decision = '2';
                        break;
                default:

                        cout << "[ОШИБКА] Неверное значение" << endl;
                        cout << "Желаете продолжить?" << endl;
                        cout << "1| Да" << endl;
                        cout << "2| Нет" << endl;
                        cout << "Ваше решение: ";
                        cin >> decision;
                        break;
                }
        }
        *this = value;
}
```

Main.cpp

```
#include "interface.cpp"

void main()
{
        setlocale(LC_ALL, "rus");
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
        Interface<bool> design;
        design.mainMenu();
}
```

Trains.cpp

```
#include "Trains.h"

Trains::Trains()
{
        hours = 0;
}
Trains::Trains(const Trains& other)
{
        hours = other.hours;
}
Trains::Trains(string nameItem, string typeItem, int costItem, int countItem, int hours) :
Item(nameItem, typeItem, costItem, countItem)
{
        this->hours = hours;
}
Trains::~Trains() = default;
void Trains::setData()
{
        string temp;
        Item::setData();
        cin >> temp;
        while (!isFigure(temp))
        {
                cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
                cout << "Введите время отправления: ";
                cin >> temp;
        }
        this->hours = stoi(temp);
}
```

```
void Trains::showHeader()
{
	Item::showHeader();
	cout << setw(18) << "Время отправления" << "|";
}
void Trains::showData()
{
	Item::showData();
	cout << setw(18) << hours << "|";
}

void Trains::editData()
{
	Item::editData();
	cout << "5| Время отправления" << endl;
}
void Trains::showLabel()
{
	Item::showLabel();
	cout << "5| Время отправления" << endl;

}

Trains Trains::operator=(const Trains& obj)
{
	this->nameItem = obj.nameItem;
	this->seats = obj.seats;
	this->price = obj.price;
	this->date = obj.date;
	this->hours = obj.hours;
	return *this;
}
bool Trains::operator==(const Trains& obj)
{
	bool flag = true;
	if (obj.nameItem != "")
	{
		if (this->nameItem == obj.nameItem)
		{
			flag = true;
		}
		else { return false; }
	}
	if (obj.seats != "")
	{
		if (this->seats == obj.seats)
		{
			flag = true;
		}
		else { return false; }
	}
	if (obj.price != NULL)
	{
		if (this->price == obj.price)
		{
			flag = true;
		}
		else { return false; }
	}
```

```
	if (obj.date != NULL)
	{
		if (this->date == obj.date)
		{
			flag = true;
		}
		else { return false; }
	}
	if (obj.hours != NULL)
	{
		if (this->hours == obj.hours)
		{
			flag = true;
		}
		else { return false; }
	}
	return flag;
}

ostream& operator<<(ostream& sout, Trains& obj)
{
	sout << "|" << setw(21) << left << obj.nameItem << "|" << setw(16) << obj.seats <<
"|" << setw(6) << obj.price << "|" << setw(7) << obj.date << "|" << setw(18) << obj.hours;
	return sout;
}
istream& operator >> (istream& sin, Trains& obj)
{
	string temp;
	cout << "Введите направление: ";
	sin >> obj.nameItem;
	cout << "Введите к-во мест: ";
	sin >> obj.seats;
	cout << "Введите цену: ";
	sin >> temp;
	while (!isFigure(temp))
	{
		cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
		cout << "Введите год: ";
		cin >> temp;
	}
	obj.price = stoi(temp);
	cout << "Введите дату: ";
	sin >> temp;
	while (!isFigure(temp))
	{
		cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
		cout << "Введите дату: ";
		cin >> temp;
	}
	obj.date = stoi(temp);
	cout << "Введите максимальную скорость: ";
	sin >> temp;
	while (!isFigure(temp))
	{
		cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
		cout << "Введите максимальную скорость: ";
		cin >> temp;
	}
	obj.hours = stoi(temp);
	return sin;
```

```
}
ofstream& operator<<(ofstream& fout, Trains& obj)
{
	fout << obj.nameItem << " " << obj.seats << " " << obj.price << " " << obj.date << "
" << obj.hours;
	return fout;
}
ifstream& operator >> (ifstream& fin, Trains& obj)
{
	fin >> obj.nameItem;
	fin >> obj.seats;
	fin >> obj.price;
	fin >> obj.date;
	fin >> obj.hours;
	return fin;
}

bool compareTime(const Trains* obj, const Trains* obj2, int trigger)
{
	switch (trigger)
	{
	case 1:
		return obj->date < obj2->date;
		break;
	case 2:
		return obj->date > obj2->date;
		break;
	}
	return false;
}
```

VipTrain.cpp

```
#include "vipTrain.h"

vipTrain::vipTrain() : Item()
{
	vipCount = "0";
}
vipTrain::vipTrain(const vipTrain& other) : Item(other)
{
	vipCount = other.vipCount;
}
vipTrain::vipTrain(string path, string seats, int price, int date, string vipCount) :
Item(path, seats, price, date)
{
	this->vipCount = vipCount;
}
vipTrain::~vipTrain() = default;

void vipTrain::setData()
{
	Item::setData();

	while (true)
	{
		cout << "Введите количество ВИП мест: "; cin >> vipCount;
		if (!isFigure(vipCount))
		{
			cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
			continue;
```

```
			}
			else
				break;
		}
}

void vipTrain::showData()
{
		Item::showData();
		cout << setw(10) << vipCount << "|";
}

void vipTrain::showHeader()
{
		Item::showHeader();
		cout << setw(10) << "ВИП места" << "|";
}

void vipTrain::showLabel()
{
		Item::showLabel();
		cout << "5| ВИП места" << endl;
}

void vipTrain::editData()
{
		Item::editData();
		cout << "5| ВИП места" << endl;
}

ostream& operator<<(ostream& sout, vipTrain& obj)
{
		sout << "|" << setw(21) << left << obj.nameItem << "|" << setw(16) << obj.seats <<
"|" << setw(6) << obj.price << "|" << setw(7) << obj.date << "|" << setw(10) <<
obj.vipCount << "|";
		return sout;
}

istream& operator >> (istream& sin, vipTrain& obj)
{
		string temp;
		cout << "Введите направление: ";
		sin >> obj.nameItem;
		cout << "Введите к-во мест: ";
		sin >> obj.seats;
		cout << "Введите цену: ";
		sin >> temp;
		while (!isFigure(temp))
		{
			cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
			cout << "Введите год: ";
			cin >> temp;
		}
		obj.price = stoi(temp);
		cout << "Введите рейтинг: ";
		sin >> temp;
		while (!isFigure(temp))
		{
			cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
			cout << "Введите рейтинг: ";
```

```
			cin >> temp;
		}
		obj.date = stoi(temp);

		cout << "Введите количество ВИП мест: ";
		sin >> temp;
		while (!isFigure(temp))
		{
			cout << "[ОШИБКА] Имеются запрещенные символы" << endl;
			cout << "Введите количество ВИП мест: ";
			cin >> temp;
		}
		obj.vipCount = temp;
		return sin;
}

ofstream& operator<<(ofstream& fout, vipTrain& obj)
{
		fout << obj.nameItem << " " << obj.seats << " " << obj.price << " " << obj.date << "
" << obj.vipCount;
		return fout;
}

ifstream& operator >> (ifstream& fin, vipTrain& obj)
{
		fin >> obj.nameItem;
		fin >> obj.seats;
		fin >> obj.price;
		fin >> obj.date;
		fin >> obj.vipCount;
		return fin;
}

vipTrain vipTrain::operator=(const vipTrain& obj)
{
		this->nameItem = obj.nameItem;
		this->seats = obj.seats;
		this->price = obj.price;
		this->date = obj.date;
		this->vipCount = obj.vipCount;
		return *this;
}

bool vipTrain::operator==(const vipTrain& obj)
{
		bool flag = true;
		if (obj.nameItem != "")
		{
			if (this->nameItem == obj.nameItem)
			{
				flag = true;
			}
			else { return false; }
		}
		if (obj.seats != "")
		{
			if (this->seats == obj.seats)
			{
				flag = true;
			}
```

```
                else { return false; }
        }
        if (obj.price != NULL)
        {
                if (this->price == obj.price)
                {
                        flag = true;
                }
                else { return false; }
        }
        if (obj.date != NULL)
        {
                if (this->date == obj.date)
                {
                        flag = true;
                }
                else { return false; }
        }
        if (obj.vipCount != "")
        {
                if (this->vipCount == obj.vipCount)
                {
                        flag = true;
                }
                else { return false; }
        }
        return flag;
}

bool compareVipSeat(const vipTrain* obj, const  vipTrain* obj2, int trigger)
{
        switch (trigger)
        {
        case 1:
                return stoi(obj->vipCount) < stoi(obj2->vipCount);
                break;
        case 2:
                return stoi(obj->vipCount) > stoi(obj2->vipCount);
                break;
        }
        return false;
}
```