МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего

образования

«Казанский национальный исследовательский технический университет

им. А.Н. Туполева-КАИ»

Институт компьютерных технологий и защиты информации

Кафедра Прикладной математики и Информатики имени Ю. В. Кожевникова

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОЙ РАБОТЕ**

**по дисциплине** «Структуры и алгоритмы обработки данных»

**Тема работы**: Реализация комбинированных структур данных

Вариант № 88.14

Выполнил:   
 студент группы 4211

Каримов А. Т

Проверил:   
 Доцент каф. ПМИ к.т.н.

Сотников С. В.

Казань 2025

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc199878405)

[1. ПОСТАНОВКА ЗАДАЧИ 4](#_Toc199878406)

[1.1 Цель работы 4](#_Toc199878407)

[1.2 Исходные данные 4](#_Toc199878408)

[1.3 Ожидаемый результат 4](#_Toc199878409)

[1.4 Требования к реализации 5](#_Toc199878410)

[2. ОПИСАНИЕ РАЗРАБОТАННЫХ КЛАССОВ. АЛГОРИТМЫ РЕАЛИЗАЦИИ ОСНОВНЫХ ОПЕРАЦИЙ 6](#_Toc199878411)

[2.1 Реализация очереди на динамическом распределении памяти 6](#_Toc199878412)

[2.2 Реализация однонаправленного упорядоченного списка на статическом распределении памяти 9](#_Toc199878413)

[2.3 Краткое описание разработанных классов 14](#_Toc199878414)

[3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 15](#_Toc199878415)

[4. РУКОВОДСТВО ПРОГРАММИСТА 19](#_Toc199878416)

[4.1.Описание структуры программы 19](#_Toc199878417)

[4.2. Описание разработанных классов и подпрограмм 19](#_Toc199878418)

[4.3. Описание структуры внешнего файла 22](#_Toc199878419)

[ВЫВОД 23](#_Toc199878420)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 24](#_Toc199878421)

[ПРИЛОЖЕНИЕ. ЛИСТИНГИ ВСЕХ ФАЙЛОВ ПРОЕКТА 25](#_Toc199878422)

# ВВЕДЕНИЕ

В данной курсовой работе рассматривается реализация структуры данных “линейный статический упорядоченный однонаправленный список динамических очередей”.

Программа реализована на основе принципов объектно-ориентированного программирования (ООП) с использованием языка программирования С++ и среды разработки Visual Studio Community.

В пояснительной записке к курсовой работе описана постановка задачи, которую необходимо выполнить, алгоритм реализации программы, руководство пользователя и руководство программиста, включающие инструкции по работе с программой, объяснение программной реализации основных алгоритмов для возможности их совершенствования и последующего использования в других проектах.

В приложении к пояснительной записке включены листинги исходных кодов всех файлов проекта.

# 1. ПОСТАНОВКА ЗАДАЧИ

## 1.1 Цель работы

Программная реализация комбинированной структуры данных –статический упорядоченный однонаправленный список динамических очередей – на основе объектно-ориентированного подхода, повторение и закрепление полученных знаний в ходе изучения курсов «Структуры и алгоритмы обработки данных» и «Объектно-ориентированное программирование», выработка навыков владения языком программирования.

## 1.2 Исходные данные

Вариант 88 задания на курсовую работу по ООП и САОД: *статический упорядоченный однонаправленный список динамических очередей*.   
Вариант 14 информационного наполнения контейнера: *Горэлектротранс – композиция трампарков (номер), Трампарк – композиция трамваев (марка, год выпуска).* Главная структура (в скобках – возможное свойство) – это набор подчиненных структур (в скобках – возможное свойство, связываемое с каждой структурой); подчиненная структура – это набор элементов нижнего уровня (в скобках – два возможных информационных свойства, для хранения и обработки которых должны использоваться соответствующие объекты).

Возможные языки программирования: Delphi Pascal, C, C++.

## 1.3 Ожидаемый результат

Программно-реализованная комбинированная структура данных (согласно варианту 88 задания на курсовую работу и варианту 14 наполнения контейнера) с использованием объектного подхода и языка программирования C++, согласно всем требованиям к реализации, описанным в следующем подпункте.

Приобретение и усвоение новых навыков программирования и работы со структурами данных. Закрепление умений применения объектно-ориентированного подхода в программировании. Улучшение навыков работы с языком программирования C++.

## 1.4 Требования к реализации

1. Реализация комбинированной структуры на основе объектного подхода;

2. Реализация всех необходимых операций (добавление и удаление в основной и присоединенной структурах, поиск в списке);

3. Возможность сохранения всей структуры по внешнем файле (текстовом или XML) с обратной загрузкой;

4. Реализация структуры для хранения и обработки данных конкретной информационной задачи;

5. Именование типов, структур и их полей, классов и их свойств, методов в соответствии с конкретной информационной задачей.

# 2. ОПИСАНИЕ РАЗРАБОТАННЫХ КЛАССОВ. АЛГОРИТМЫ РЕАЛИЗАЦИИ ОСНОВНЫХ ОПЕРАЦИЙ

## 2.1 Реализация очереди на динамическом распределении памяти

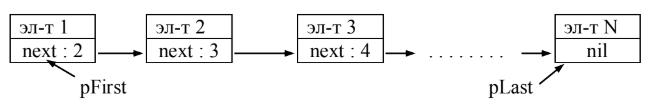
Очередь – это линейная структура данных, в которую элементы добавляются с одного конца (конец очереди), а удаляются - с другого (начало очереди). Очередь работает по принципу “элемент, помещенный в очередь первым, извлечен будет тоже первым”. Иногда этот принцип обозначается сокращением FIFO (First In – First Out, т.е. первым зашел – первым вышел). Элементами стеков и очередей могут быть любые однотипные данные. В простейшем случае – целые числа, чаще всего – записи заранее определенной структуры. Очереди очень широко используются в системных программах, в частности – в операционных системах и компиляторах.

Программная реализация очереди возможна двумя способами:

* статически с помощью массива
* динамически с помощью механизма указателей

В курсовой работе был использован второй способ. Для динамической реализации структуры Очередь необходимо, чтобы каждый элемент очереди имел ссылку на следующий за ним элемент, поэтому элемент очереди объявляется как запись с двумя полями – информационное поле и связующее поле. Но для реализации операций с очередью необходимы уже две переменные: указатель pFirst на начало очереди и указатель pLast на конец очереди.

Приведенная ниже схема элементов очереди отражает логический порядок следования элементов, физически же элементы могут находиться любых свободных областях памяти.



Для прохода по очереди от первого реального элемента к последнему необходимо:

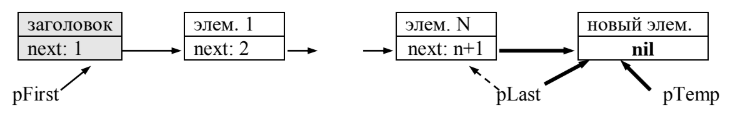
* ввести вспомогательную ссылочную переменную pTemp
* установить pTemp в адрес первого реального элемента: pTemp :=

pFirst^.next

* организовать цикл по условию достижения конца очереди
* в цикле обработать очередной элемент с помощью указателя pTemp и
* изменить этот указатель: pTemp := pTemp^.next

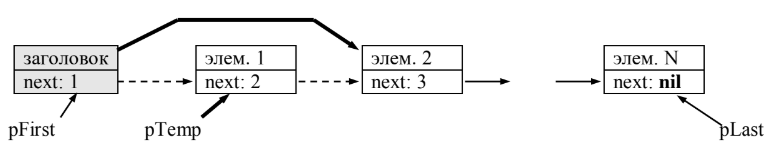
Добавление элемента в конец очереди выполняется следующим образом:

* выделить память для нового элемента с помощью стандартной функции New и вспомогательной ссылочной переменной pTemp:
* заполнить поля нового элемента, в частности в связующую часть установить значение nil: pTemp^.next := nil
* изменить связующую часть бывшего последнего элемента таким образом, чтобы она адресовала новый добавленный элемент: pLast^.next := pTemp;
* изменить значение указателя pLast так, чтобы он указывал новый последний элемент: pLast := pTemp;



Удаление элемента из начала очереди (но после заголовка!) выполняется следующим образом:

* адресуем удаляемый элемент с помощью вспомогательной переменной pTemp : pTemp := pFirst^.next;
* x изменить связующую часть заголовка так, чтобы она указывала на второй элемент очереди, который теперь должен стать первым: pFirst^.next := pTemp^.next
* если после удаления в списке не остаётся реальных элементов, то необходимо изменить указатель pLast: pLast := pFirst
* обработать удаленный элемент, например - освободить занимаемую им память с помощью стандартной подпрограммы Dispose (pTemp) или включить его во вспомогательную очередь удаленных элементов



## 2.2 Реализация однонаправленного упорядоченного списка на статическом распределении памяти

Линейный список – это набор связанных однотипных элементов, в котором каждый элемент каким-то образом определяет следующий за ним элемент. Добавление нового элемента возможно в любом месте списка, также можно удалить любой элемент списка.

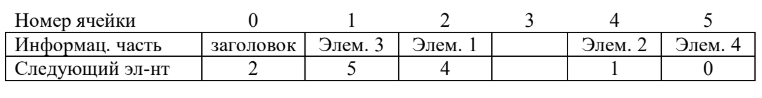
Возможна статическая и динамическая реализация списков. Динамическая реализация линейного списка основана на динамическом выделении и освобождении памяти для элементов списка. Логическая последовательность элементов списка создается ссылочными переменными с адресами последующих элементов.

Для удобства реализации, список всегда содержит хотя бы один элемент-заголовок с адресом первого реального элемента списка. Это позволяет унифицировать процедуры добавления и удаления крайних элементов и устранить некоторые проверки. Адрес элемента-заголовка задается переменной-указателем “pHead”. Эта переменная устанавливается при первоначальном создании списка и в дальнейшем не изменяется. Для реализации основных действий используются вспомогательные ссылочные переменные.

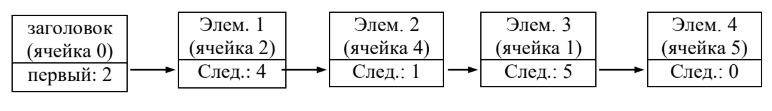
Одна из разновидностей линейного списка – упорядоченный список. В нем элементы выстраиваются в соответствии с заданным порядком, например – целые числа по возрастанию, текстовые строки по алфавиту. Для таких списков изменяется процедура добавления – новый элемент должен вставляться в соответствующее место для сохранения порядка элементов. Например, если порядок элементов определяется целыми числами по возрастанию, то при поиске подходящего места надо найти первый элемент, больший заданного и выполнить вставку ПЕРЕД этим элементом.

Один из способов реализации списка – это на основе массива, используемого принцип указателей (но БЕЗ динамического распределения памяти). В этом случае каждый элемент списка (кроме последнего) должен содержать номер ячейки массива, в которой находится следующий за ним элемент. Это позволяет РАЗЛИЧАТЬ физический и логический порядок следования элементов в списке. Удобно (но НЕ обязательно) в начале массива ввести фиктивный элемент-заголовок, который всегда занимает нулевую ячейку массива, никогда не удаляется и указывает индекс первого реального элемента списка. В этом случае последний элемент списка (где бы он в массиве не располагался) должен в связующей части иметь некоторое специальное значение-признак, например – индекс 0.

Схема физического размещения элементов списка в массиве:



Соответствующая схема логического следования элементов списка:



Рассмотрим реализацию основных списковых операций:

**Создание пустого списка:**

* выделяется память под заголовок, адресуемая указателем pHead
* оба ссылочных поля заголовка устанавливаются в адрес самого заголовка: pHead^.left := pHead; pHead^.right := pHead;

**Проход по списку:**

* ввести вспомогательную переменную Current для отслеживания текущего элемента списка и установить Current := StatList [ 0 ].Next;
* организовать цикл по условию Current = 0, внутри которого обработать текущий элемент StatList [ Current ].Inf и изменить указатель Current на следующий элемент: Current := StatList [ Current ].Next

**Поиск элемента аналогичен** проходу, но может заканчиваться до достижения конца списка:

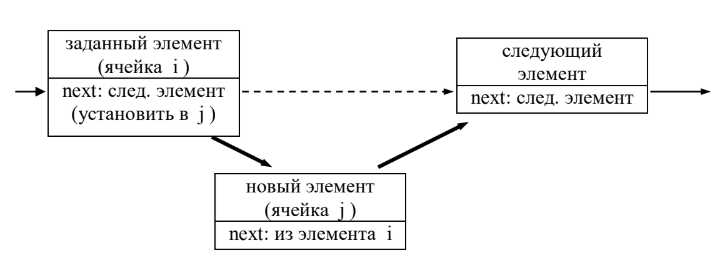
Current := StatList [ 0 ].Next;

While (Current <> 0) and (StatList [ Current ].Inf <> ‘значение’ do

Current := StatList [ Current ].Next;

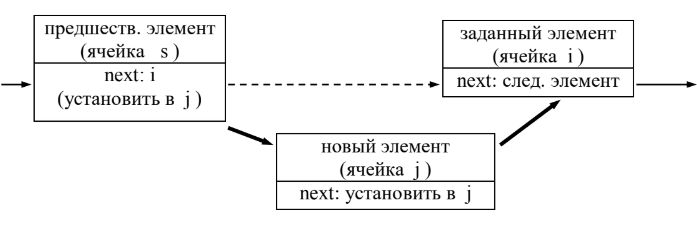
If Current = 0 then ‘поиск неудачен’ else ‘элемент найден’;

**Добавление элемента после заданного:**

* проверка возможности добавления с помощью счетчика текущего числа элементов в списке
* определение каким-то образом элемента, после которого надо добавить новый элемент (например – запрос у пользователя)
* поиск этого элемента в списке; пусть его индекс есть i
* определение номера свободной ячейки массива для размещения нового элемента (методы определения будут рассмотрены ниже); пусть этот номер равен j
* формирование связующей части нового элемента, т.е. занесение туда номера ячейки из связующей части элемента i : StatList [ j ].next := StatList [ i ].next;
* изменение связующей части элемента i на номер j: StatList [ i ].next := j;
* занесение данных в информационнуючасть нового элемента StatList[j].inf; 

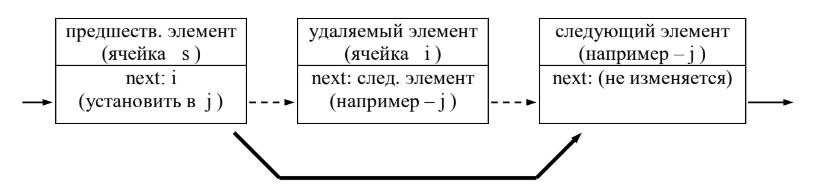
Алгоритм добавления элемента перед заданным включает следующие шаги:

* проверка возможности добавления с помощью счетчика текущего числа элементов в списке
* определение каким-то образом элемента, перед которым надо добавить новый элемент (например – запрос у пользователя)
* поиск этого элемента в списке с одновременным отслеживанием элемента-предшественника; пусть индекс заданного элемента есть i, а индекс его предшественника - s
* определение номера свободной ячейки массива для размещения нового элемента (методы определения будут рассмотрены ниже); пусть этот номер равен j
* формирование связующей части нового элемента, т.е. занесение туда индекса i : StatList [ j ].next := i;
* изменение связующей части элемента-предшественника с индекса i на индекс j: StatList [ s ].next := j;
* занесение данных в информационную часть нового элемента StatList[j].inf;



**Удаление заданного элемента** (естественно, в случае его наличия в списке) также требует изменения связующей части у элемента-предшественника. Это изменение позволяет “обойти” удаляемый элемент и тем самым исключить его из списка. Необходимые шаги:

* определение каким-то образом удаляемого элемента (например – запрос у пользователя)
* поиск удаляемого элемента в списке с одновременным отслеживанием элемента-предшественника; пусть индекс удаляемого элемента есть i, а индекс его предшественника - s
* изменение связующей части элемента-предшественника с индекса i на индекс-значение связующей части удаляемого элемента i: StatList [s].next:= StatList [ i ].next;
* обработка удаляемого элемента (например – вывод информационной части)
* включение удаленного элемента во вспомогательный список без его уничтожения или освобождение ячейки i с включением ее в список свободных ячеек (методы поддержки свободной памяти рассматриваются ниже)



## 2.3 Краткое описание разработанных классов

При выполнении курсовой работы было создано 4 классов и одна главная подпрограмма с функцией main(). Через эту подпрограмму осуществляется диалог с пользователем. Краткое описание разработанных классов (более подробно см. в пункте «Руководство программиста»):

**Класс Tram.** Является классом, содержащим данные о трамвае (марка, год выпуска) и методы взаимодействия с этими данными (Get/Set методы). Предоставляет функции преобразования данных в строку для вывода и сохранения.

**Класс TramPark.** Реализует динамическую очередь трамваев (объектов класса Tram) с использованием стандартного контейнера queue. Содержит методы для управления очередью: добавление/удаление трамваев, поиск по марке, получение информации о первом трамвае в очереди. Также включает номер парка и методы для работы с ним.

**Класс CityTransport.** Реализует статический однонаправленный упорядоченный список трамвайных парков на основе массива фиксированного размера с системой указателей (индексов) между элементами. Каждый элемент списка содержит объект TramPark и индекс следующего элемента. Класс включает методы для управления списком: добавление/удаление парков с сохранением порядка, поиск парка по номеру, а также методы для вывода данных и работы со списком свободных ячеек.

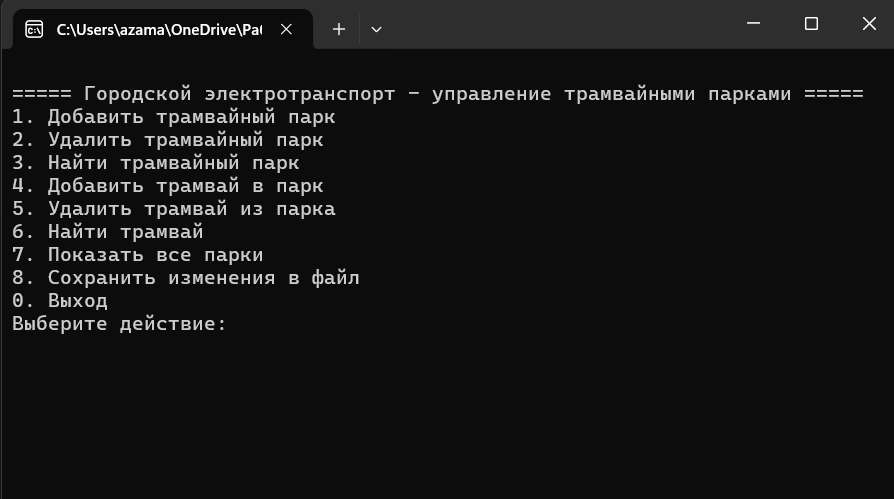
**Класс Storage**. Осуществляет взаимодействие парков и текстового файла. Получив ссылку на объект парк, считывает данные из файла и дополняет его.

# 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данная программа позволяет управлять структурой трамвайного депо, включая создание парков с уникальными номерами (автоматически сортируются), каждый из которых содержит очередь трамваев (с указанием марки и года выпуска). Поддерживаются операции добавления/удаления трамваев, поиск по марке. Также программа поддерживает работу с файлом: при каждом запуске происходит поиск текстового файла “transport\_data.txt” и ввод данных из него в таблицу (если такой файл существует и найден, иначе создается пустое меню)

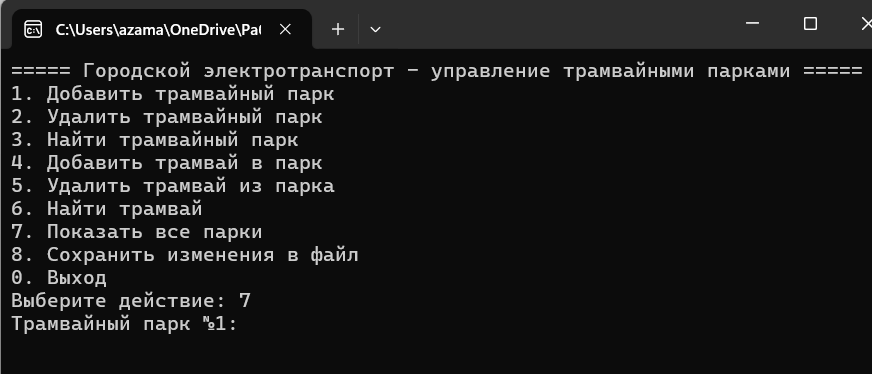
При завершении работы программы Вам будет предложено сохранить изменения в файле (сделать перезапись файла), но будьте осторожны: файл будет полностью перезаписан без возможности вернуть его в первоначальное состояние.

Если при запуске файл “transport\_data.txt” не был найден на момент запуска приложения, то Вам будет показано предупреждение. Перед вами окажется данное меню:



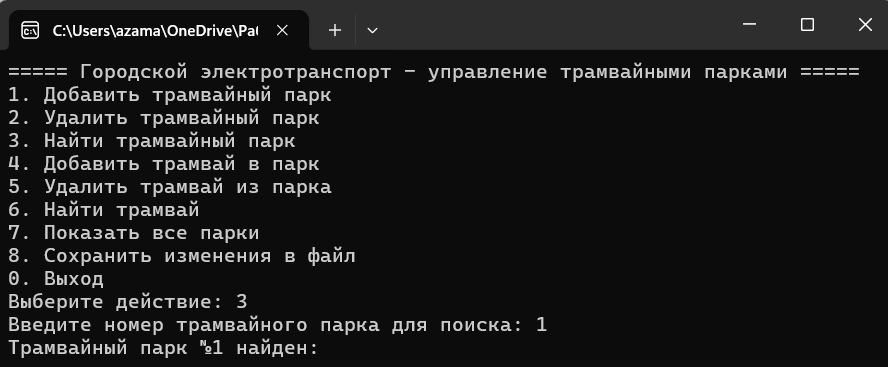
Выберите любую команду из предложенного списка, введите её и нажмите Enter

Выбрав **первый пункт (Добавить трамвайный парк)**, Вам будет предложено ввести номер нового трамвайного парка. После введения и нажатия клавиши Enter добавится.

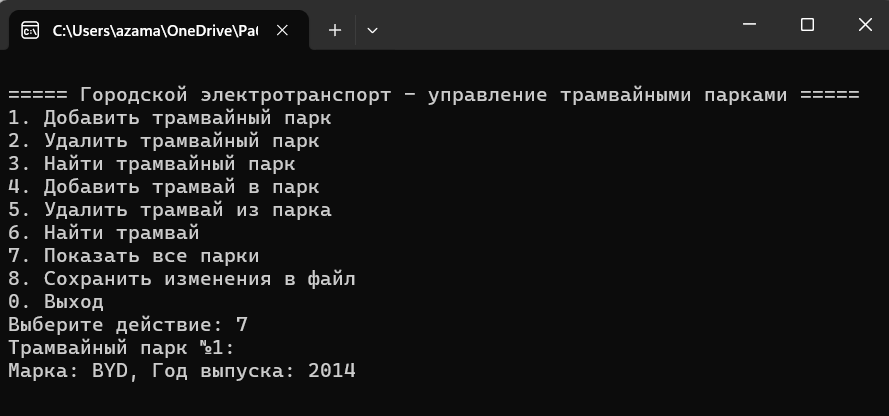


Если вы выберите **второй пункт** главного меню (**Удалить трамвайный парк**), то Вам будет предложено ввести номер удаляемого парка. После введения названия и нажатия клавиши Enter будет произведено удаление.

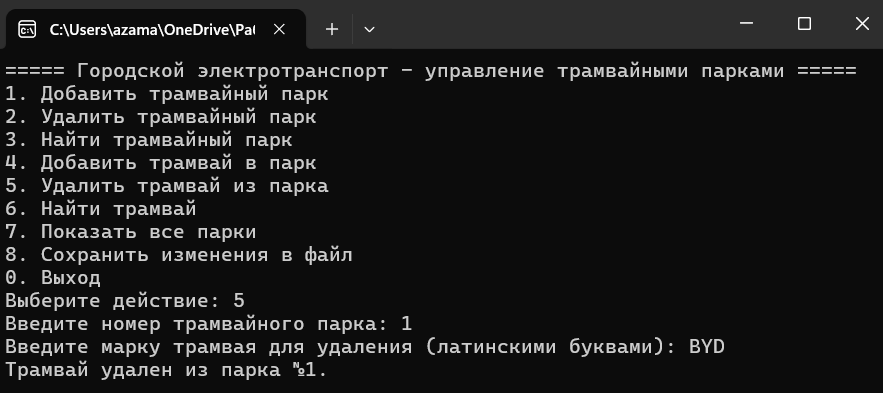
Если вы выберите **третий пункт** в главном менюпрограммы (**Найти трамвайный парк**), то Вам будет предложено ввести номер парка, которое вы хотите найти. После введения названия и нажатия клавиши Enter будет произведен поиск и последующее отображение.



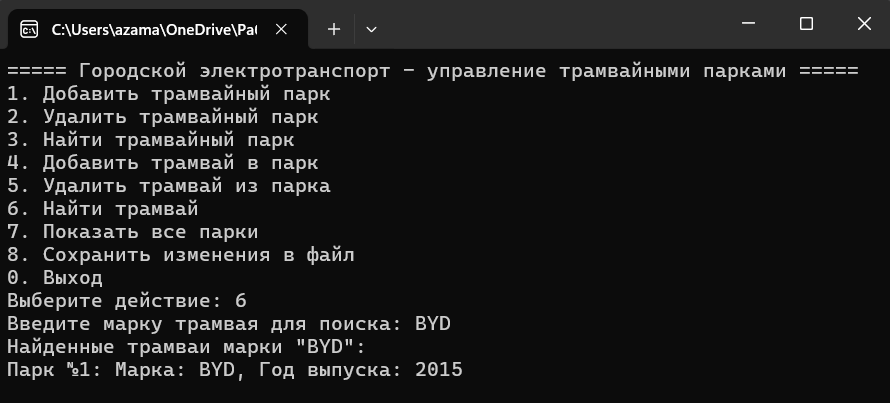
Если вы выберите **четвертый пункт** в главном менюпрограммы (**Добавить трамвай в парк)**, то Вам будет предложено ввести номер парка, в который вы хотите добавить новый трамвай. После введения названия и года и нажатия клавиши Enter в будет добавлен трамвай.



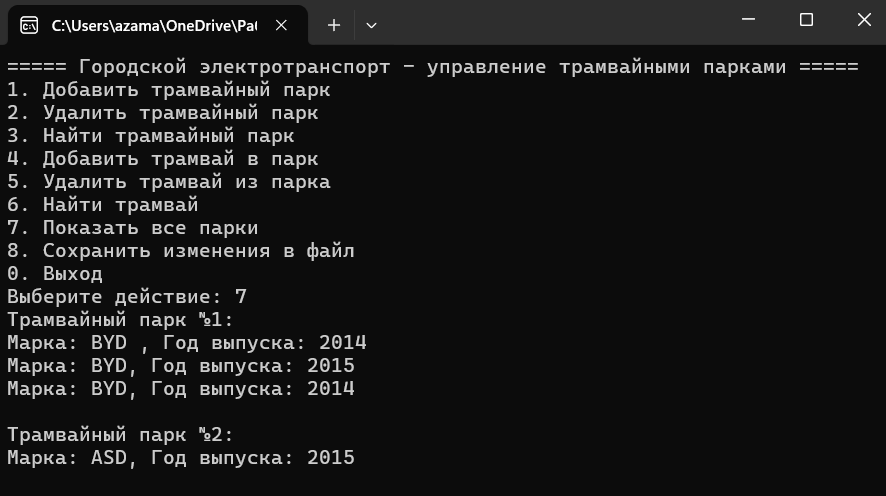
Если вы выберите **пятый пункт** главного меню (**Удалить трамвай из парк**), то Вам будет предложено ввести номер парка, а после название удаляемого трамвая.



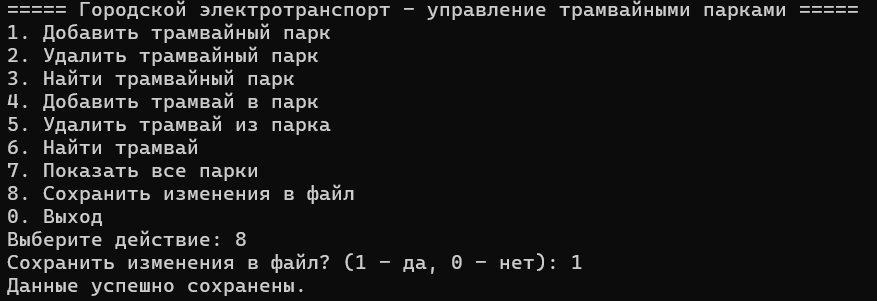
Если вы выберите **шестой пункт** в главном менюпрограммы (**Найти трамвай**), то Вам будет предложено ввести название трамвая, которое вы хотите найти.



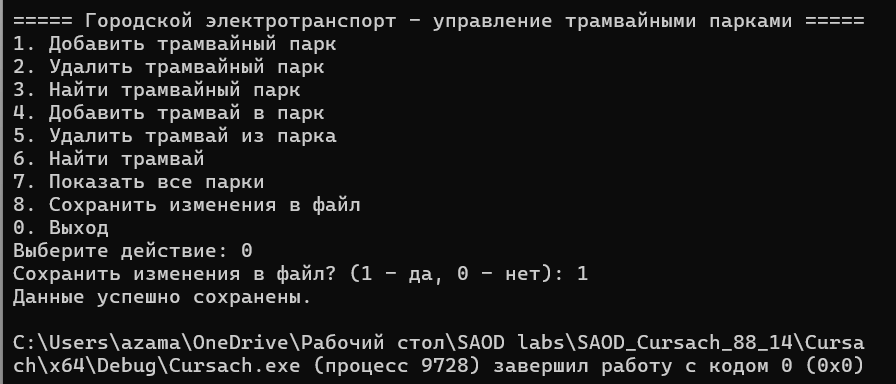
Выбрав **седьмой пункт** главного меню **(Показать все парки)**, будет отображены все парки со своими трамваями.



Выбрав **восьмой** **пункт** главного меню **(Сохранить изменения в файл)**, будут предложено сохранить изменения в файл.



Выбрав **пункт 0** главного меню **(ВЫХОД)**, Вам будет предложено сохранить изменения в файле или выход без сохранения. Будьте осторожны: файл будет полностью перезаписан без возможности вернуть его в первоначальное состояние. После выбора желаемого варианта (ДА/НЕТ) программа будет завершена.



Если при сохранении файл не будет найден, то создастся новый файл, и данные будут сохранены. Осторожно: при сохранении таблицы старая версия будет полностью утеряна!

# 4. РУКОВОДСТВО ПРОГРАММИСТА

### 4.1.Описание структуры программы

Данный проект был создан для демонстрации работы комбинированной структуры «линейный статический упорядоченный однонаправленный список динамических очередей» с использованием объектно-ориентированного программирования (ООП). Для создания программы был выбран язык C++ с поддержкой ООП. В качестве среды разработки было выбрано приложение Visual Studio Community (2022).

Главный структура - однонаправленный упорядоченный линейный список на статическом распределении памяти. Подструктура – очередь на динамическом распределении памяти.

В ходе разработки было создано 4 классов, каждый из которых был помещен в отдельный файл. Также был создан отдельный файл с главной подпрограммой, содержащей функцию «int main()».

CityTranspot

Поля класса:

…

TramPark\*

head

TramPark

Поля класса:

…

Tram\*

next

Tram

Поля класса:

…

### 4.2. Описание разработанных классов и подпрограмм

**Класс Tram:**

|  |  |
| --- | --- |
| **Поле класса** | **Описание поля** |
| string model | Марка трамвая |
| int year | Год выпуска трамвая |
| **Метод класса** | **Описание метода** |
| Tram() | Конструктор по умолчанию |
| Tram(string model, int year) | Конструктор с параметрами |
| void Set(string model, int year) | Метод ввода данных в структуру объекта |
| string GetModel() | Возвращает номер поля “model” из структуры |
| int GetYear() | Возвращает номер поля “year” из структуры |
| Void Clear() | Метод очистки структуры команды |
| ~ Tram() | Деструктор объекта класса Tram |

**Класс TramPark:**

|  |  |
| --- | --- |
| **Поле класса** | **Описание поля** |
| Node\* head | Указатель на начало очереди |
| Node\* rear | Указатель на конец очереди |
| int parkNumber | Уникальный номер трамвайного парка. |
| Tram\* data | Указатель на объект трамвая |
| Node\* next | Указатель на следующий узел |
| **Метод класса** | **Описание метода** |
| TramPark() | Конструктор по умолчанию |
| TramPark(int number) | Конструктор с параметром |
| bool Enqueue(Tram\* tram) | Метод добавления трамвая в очередь. |
| bool Dequeue(string model) | Удаляет трамвай из очереди |
| Tram\* FindTram(const string& model) | Поиск модели по заданному значению. |
| void ShowTrams() | Вывод на экран всех трамваев |
| string GetString() | Формирует текстовую строку для записи данных в файл |
| ~ TramPark() | Деструктор объекта класса TramPark |

**Класс CityTransport:**

|  |  |
| --- | --- |
| **Поле класса** | **Описание поля** |
| ParkNode parksArray[n] | Статический массив структур ParkNode |
| int head | Индекс первого элемента в списке парков. |
| int freeList | Индекс первой свободной ячейки в parksArray |
| TramPark\* park | Указатель на объект класса TramPark |
| int next | Индекс следующей свободной ячейки |
| bool used | Логический флаг для проверки занятости |
| **Метод класса** | **Описание метода** |
| CityTransport() | Конструктор объекта класса CityTransport. При его вызове создается заголовочный элемент |
| AddPark(int parkNumber) | Добавление нового парка с сохранением порядка |
| RemovePark(int parkNumber) | Удаление парка с заданным номером |
| FindPark(int parkNumber) | Поиск парка по содержимому |
| ShowParks() | Метод вывода всех парков на экран |
| FindTram(string model) | Метод поиска трамвая по всем паркам |
| void InitFreeList() | Инициализация списка свободных узлов в массиве |
| int AllocateNode() | Выделение свободного узла для нового парка |
| void FreeNode(int index) | Освобождение узла с заданным индексом |
| ~ CityTransport() | Деструктор объекта класса CityTransport |

**Класс Storage:**

|  |  |
| --- | --- |
| **Поле класса** | **Описание поля** |
| string filename | Путь к файлу для хранения данных |
| **Метод класса** | **Описание метода** |
| Storage(string \_filename) | Конструктор c указанием пути к файлу |
| Bool SaveData(CityTranspot\* transport) | Сохраняет данные в файл |
| bool LoadData(CityTranspot\* transport) | Загружает данные из файла |
| ~Storage() | деструктор объекта класса Storage |

## 4.3. Описание структуры внешнего файла

Информационная модель:

* **Горэлектротранс** — содержит список трамвайных парков.
* **Трамвайный парк** — содержит номер и список трамваев.
* **Трамвай** — содержит марку и год выпуска бензина.

Формат файла:  
<Номер ТП 1> <Марка трамвая> <Год выпуска трамвая> ... <Марка трамвая> <Год выпуска трамвая>

…

<Номер ТП N> <Марка трамвая> <Год выпуска трамвая> ... <Марка трамвая> <Год выпуска трамвая>

## ВЫВОД

В результате выполнения данной курсовой работы была создана программа, реализующая структуру данных *статический упорядоченный однонаправленный список динамических очередей,*  используя принципы объектно-ориентированного программирования. Были повторены и закреплены знания, полученные в ходе изучения курсов «Структуры и алгоритмы обработки данных» и «Объектно-ориентированное программирование», а также были выработаны новые навыки владения языком программирования С++.

На основе исходных данных выполнены все требования к реализации программы, получен ожидаемый результат.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Козин А. Н. Структуры и алгоритмы обработки данных. Учебно-методическое пособие. Издательство: Татарский институт содействия бизнесу, 2003.

2. Козин А. Н. Объектно-ориентированное программирование. Учебно-методическое пособие. Издательство: Академия Управления «ТИСБИ», 2006.

# ПРИЛОЖЕНИЕ. ЛИСТИНГИ ВСЕХ ФАЙЛОВ ПРОЕКТА

main.cpp – главная программа

#include <iostream>

#include "CityTransport.h"

#include "Storage.h"

#include <io.h>

#include <fcntl.h>

#include <Windows.h>

#ifdef max

#undef max

#endif

using namespace std;

CityTransport\* transport = NULL;

Storage file("transport\_data.txt");

// Обработка ошибок

int failure() {

int a;

while (!(cin >> a) || (cin.peek() != '\n')) {

cin.clear();

while (cin.get() != '\n');

cout << "\nОшибка ввода. Повторите ввод: ";

}

return a;

}

// Добавление парка

void AddPark() {

int parkNumber;

cout << "Введите номер трамвайного парка: ";

parkNumber = failure();

transport->AddPark(parkNumber);

}

// Удаление парка

void RemovePark() {

int parkNumber;

cout << "Введите номер трамвайного парка для удаления: ";

parkNumber = failure();

if (transport->RemovePark(parkNumber)) {

cout << "Трамвайный парк №" << parkNumber << " удален.\n";

}

else {

cout << "Трамвайный парк не найден.\n";

}

}

// Поиск парка

void FindPark() {

int parkNumber;

cout << "Введите номер трамвайного парка для поиска: ";

parkNumber = failure();

TramPark\* park = transport->FindPark(parkNumber);

if (park != NULL) {

cout << "Трамвайный парк №" << parkNumber << " найден:\n";

park->ShowTrams();

}

else {

cout << "Трамвайный парк не найден.\n";

}

}

// Добавление трамвая

void AddTram() {

int parkNumber;

cout << "Введите номер трамвайного парка: ";

parkNumber = failure();

TramPark\* park = transport->FindPark(parkNumber);

if (park == NULL) {

cout << "Трамвайный парк не найден.\n";

return;

}

string model;

int year;

cout << "Введите марку трамвая: ";

cin.clear(); cin.ignore(32767, '\n'); getline(cin, model);

cout << "Введите год выпуска: ";

year = failure();

Tram\* tram = new Tram();

tram->Set(model, year);

if (park->Enqueue(tram)) {

cout << "Трамвай добавлен в парк №" << parkNumber << ".\n";

}

else {

cout << "Не удалось добавить трамвай.\n";

delete tram;

}

}

// Удаление трамвая

void RemoveTram() {

int parkNumber;

cout << "Введите номер трамвайного парка: ";

parkNumber = failure();

TramPark\* park = transport->FindPark(parkNumber);

if (park == NULL) {

cout << "Трамвайный парк не найден.\n";

return;

}

string model;

cout << "Введите марку трамвая для удаления: ";

cin.clear(); cin.ignore(32767, '\n'); getline(cin, model);

if (park->RemoveTram(model)) {

cout << "Трамвай удален из парка №" << parkNumber << ".\n";

}

else {

cout << "Трамвай не найден.\n";

}

}

// Поиск трамвая во всех парках

void FindTram() {

string model;

cout << "Введите марку трамвая для поиска: ";

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

getline(cin, model);

transport->FindTram(model);

}

int main() {

setlocale(LC\_ALL, "Russian");

transport = new CityTransport();

if (!file.LoadData(transport)) {

cout << "Не удалось загрузить данные из файла. Начните с пустой базы.\n";

}

int choice;

while (true) {

cout << "\n===== Городской электротранспорт - управление трамвайными парками =====\n";

cout << "1. Добавить трамвайный парк\n";

cout << "2. Удалить трамвайный парк\n";

cout << "3. Найти трамвайный парк\n";

cout << "4. Добавить трамвай в парк\n";

cout << "5. Удалить трамвай из парка\n";

cout << "6. Найти трамвай\n";

cout << "7. Показать все парки\n";

cout << "8. Сохранить изменения в файл\n";

cout << "0. Выход\n";

cout << "Выберите действие: ";

choice = failure();

switch (choice) {

case 1: AddPark(); break;

case 2: RemovePark(); break;

case 3: FindPark(); break;

case 4: AddTram(); break;

case 5: RemoveTram(); break;

case 6: FindTram(); break;

case 7: transport->ShowParks(); break;

case 8: {

cout << "Сохранить изменения в файл? (1 - да, 0 - нет): ";

int save = failure();

if (save == 1) {

if (file.SaveData(transport)) {

cout << "Данные успешно сохранены.\n";

}

else {

cout << "Ошибка при сохранении данных.\n";

}

}

break;

}

case 0: {

cout << "Сохранить изменения в файл? (1 - да, 0 - нет): ";

int save = failure();

if (save == 1) {

if (file.SaveData(transport)) {

cout << "Данные успешно сохранены.\n";

}

else {

cout << "Ошибка при сохранении данных.\n";

}

}

delete transport;

return 0;

}

default:

cout << "Неверный выбор. Попробуйте снова.\n";

}

}

}

Storage.h – класс для работы с файлом (чтение и запись)

#pragma once

#include "CityTransport.h"

#include <fstream>

#include <sstream>

class Storage

{

private:

string filename;

public:

Storage(string \_filename) {

filename = \_filename;

}

bool SaveData(CityTransport\* transport) {

ofstream file(filename);

if (!file.is\_open()) return false;

file << transport->GetString();

file.close();

return true;

}

bool LoadData(CityTransport\* transport) {

ifstream file(filename);

if (!file.is\_open()) return false;

string line;

while (getline(file, line)) {

istringstream iss(line);

int parkNumber;

iss >> parkNumber;

transport->AddPark(parkNumber);

TramPark\* park = transport->FindPark(parkNumber);

string model;

int year;

while (iss >> model >> year) {

Tram\* tram = new Tram();

tram->Set(model, year);

park->Enqueue(tram);

}

}

file.close();

return true;

}

~Storage() {}

};

CityTransport.h – класс для управления парков

#pragma once

#include "TramPark.h"

#include <iostream>

#include <vector>

using namespace std;

const int MAX\_PARKS = 2; // Максимальное количество парков

class CityTransport {

private:

struct ParkNode {

int parkNumber;

TramPark\* park;

int next;

bool used;

};

ParkNode parksArray[MAX\_PARKS];

int head;

int freeList;

void InitFreeList() {

for (int i = 0; i < MAX\_PARKS - 1; i++) {

parksArray[i].next = i + 1;

parksArray[i].used = false;

}

parksArray[MAX\_PARKS - 1].next = -1;

parksArray[MAX\_PARKS - 1].used = false;

freeList = 0;

}

int AllocateNode() {

if (freeList == -1) return -1; // нет свободных ячеек

int index = freeList;

freeList = parksArray[freeList].next;

parksArray[index].used = true;

parksArray[index].next = -1;

return index;

}

void FreeNode(int index) {

parksArray[index].next = freeList;

parksArray[index].used = false;

freeList = index;

}

public:  
// конструктор

CityTransport() : head(-1) {

InitFreeList();

}

// Добавление парка

bool AddPark(int parkNumber) {

int current = head;

while (current != -1) {

if (parksArray[current].parkNumber == parkNumber) {

cout << "Ошибка: парк с номером " << parkNumber << " уже существует!\n";

return false;

}

current = parksArray[current].next;

}

if (freeList == -1) {

cout << "Ошибка: достигнуто максимальное количество парков (" << MAX\_PARKS << ")!\n";

return false;

}

int newNodeIndex = AllocateNode();

if (newNodeIndex == -1) {

cout << "Ошибка: невозможно добавить парк (внутренняя ошибка)!\n";

return false;

}

parksArray[newNodeIndex].parkNumber = parkNumber;

parksArray[newNodeIndex].park = new TramPark(parkNumber);

if (head == -1 || parksArray[head].parkNumber > parkNumber) {

parksArray[newNodeIndex].next = head;

head = newNodeIndex;

cout << "Трамвайный парк №" << parkNumber << " успешно добавлен.\n";

return true;

}

current = head;

int prev = -1;

while (current != -1 && parksArray[current].parkNumber < parkNumber) {

prev = current;

current = parksArray[current].next;

}

parksArray[newNodeIndex].next = current;

if (prev != -1) {

parksArray[prev].next = newNodeIndex;

}

cout << "Трамвайный парк №" << parkNumber << " успешно добавлен.\n";

return true;

}

// Удаление парка

bool RemovePark(int parkNumber) {

int current = head;

int prev = -1;

while (current != -1 && parksArray[current].parkNumber != parkNumber) {

prev = current;

current = parksArray[current].next;

}

if (current == -1) return false; // не найден

if (prev == -1) {

head = parksArray[current].next;

}

else {

parksArray[prev].next = parksArray[current].next;

}

// Освобождение памяти

delete parksArray[current].park;

FreeNode(current);

return true;

}

// Поиск парка

TramPark\* FindPark(int parkNumber) {

int current = head;

while (current != -1) {

if (parksArray[current].parkNumber == parkNumber) {

return parksArray[current].park;

}

current = parksArray[current].next;

}

return nullptr;

}

// Показ парков

void ShowParks() {

int current = head;

while (current != -1) {

cout << "Трамвайный парк №" << parksArray[current].parkNumber << ":" << endl;

parksArray[current].park->ShowTrams();

cout << endl;

current = parksArray[current].next;

}

}

string GetString() {

string result;

int current = head;

while (current != -1) {

result += to\_string(parksArray[current].parkNumber) + " " +

parksArray[current].park->GetString() + "\n";

current = parksArray[current].next;

}

return result;

}

// Поиск трамвая

void FindTram(const string& model) {

bool foundAny = false;

int current = head;

while (current != -1) {

TramPark\* park = parksArray[current].park;

TramPark::Node\* node = park->GetHead()->next;

vector<Tram\*> matchedTrams;

// Собираем все трамваи нужной модели в текущем парке

while (node != nullptr) {

if (node->data->GetModel() == model) {

matchedTrams.push\_back(node->data);

}

node = node->next;

}

if (!matchedTrams.empty()) {

if (!foundAny) {

cout << "Найденные трамваи марки \"" << model << "\":" << endl;

foundAny = true;

}

cout << "Парк №" << parksArray[current].parkNumber << ":\n";

for (const auto& tram : matchedTrams) {

cout << " Марка: " << tram->GetModel()

<< ", Год выпуска: " << tram->GetYear() << endl;

}

}

current = parksArray[current].next;

}

if (!foundAny) {

cout << "Трамваи марки \"" << model << "\" не найдены.\n";

}

}

// Деструктор

~CityTransport() {

int current = head;

while (current != -1) {

delete parksArray[current].park;

int next = parksArray[current].next;

parksArray[current].used = false;

current = next;

}

}

};

TramPark.h – класс со структурой парков

#pragma once

#include "Tram.h"

#include <iostream>

using namespace std;

class TramPark

{

public:

struct Node {

Tram\* data;

Node\* next;

};

Node\* head;

Node\* rear;

int parkNumber;

// конструктор

TramPark(int number) {

head = new Node();

head->data = new Tram();

head->data->Set("<HEAD>", 0);

head->next = NULL;

rear = head;

parkNumber = number;

}

// геттер

int GetParkNumber() { return parkNumber; }

bool Enqueue(Tram\* tram) {

Node\* temp = new Node();

temp->data = tram;

temp->next = NULL;

rear->next = temp;

rear = temp;

return true;

}

// Поиск трамвая в парке

Tram\* FindTram(const string& model) const {

Node\* current = head->next;

while (current != nullptr) {

if (current->data->GetModel() == model) {

return current->data;

}

current = current->next;

}

return nullptr;

}

// Удаление трамвая

bool Dequeue (string model) {

Node\* prev = head;

Node\* current = head->next;

while (current != NULL) {

if (current->data->GetModel() == model) {

prev->next = current->next;

if (current == rear) rear = prev;

delete current->data;

delete current;

return true;

}

prev = current;

current = current->next;

}

return false;

}

// геттер для head-элемента

Node\* GetHead() const {

return head;

}

// Показ трамваев

void ShowTrams() {

Node\* current = head->next;

while (current != NULL) {

cout << " Марка: " << current->data->GetModel()

<< ", Год выпуска: " << current->data->GetYear() << endl;

current = current->next;

}

}

string GetString() {

string result;

Node\* current = head->next;

while (current != NULL) {

result += current->data->GetModel() + " " + to\_string(current->data->GetYear()) + " ";

current = current->next;

}

return result;

}

// Деструктор

~TramPark() {

while (head->next != NULL) {

Node\* temp = head->next;

head->next = temp->next;

delete temp->data;

delete temp;

}

delete head->data;

delete head;

}

}

Tram.h – класс со структурой трамвая

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Tram

{

private:

struct TramData {

string model;

int year;

} theTram;

public:

// конструктор

Tram() {

theTram.model = "<НЕТ ДАННЫХ>";

theTram.year = 0;

}

// Сеттер

void Set(string \_model, int \_year) {

theTram.model = \_model;

theTram.year = \_year;

}

// Геттеры

string GetModel() { return theTram.model; }

int GetYear() { return theTram.year; }

// Очистка

void Clear() {

theTram.model = "<НЕТ ДАННЫХ>";

theTram.year = 0;

};

// Деструктор по умолчанию

~Tram() { }

};