**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**федеральное государственное бюджетное образовательное учреждение высшего профессионального 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»

на тему «Реализация комбинированных структур данных»

вариант №93.14

**Выполнил**

Студент группы 4218

Трифонов Сергей Алексеевич

**Проверил**

Доцент кафедры ПМИ

Сотников Сергей Викторович

Казань 2023

Содержание

[Глава 1. Постановка задачи 3](#_Toc136866595)

[Глава 2. Теоретическое описание разработанной структуры 4](#_Toc136866596)

[2.1. Логическая составляющая структуры 4](#_Toc136866597)

[2.2. Набор основных операций 4](#_Toc136866598)

[2.3. Алгоритмизация основных операций 5](#_Toc136866599)

[2.4. Реализация структуры 6](#_Toc136866600)

[Глава 3. Описание программного комплекса 9](#_Toc136866601)

[3.1. Описание структуры проекта 9](#_Toc136866602)

[3.2. Руководство программиста 10](#_Toc136866603)

[3.2.1. Описание разработанных классов 10](#_Toc136866604)

[3.2.2. Описание структуры внешнего файла 12](#_Toc136866605)

[3.3. Руководство пользователя 13](#_Toc136866606)

[3.3.1. Описание демонстрационного модуля 13](#_Toc136866607)

[Заключение 18](#_Toc136866608)

[Литература 19](#_Toc136866609)

[Приложения 20](#_Toc136866610)

[Приложение 1. Листинг Kurs.cpp 20](#_Toc136866611)

[Приложение 2. Листинг Company.cpp 24](#_Toc136866612)

[Приложение 3. Листинг Trampark.cpp 26](#_Toc136866613)

[Приложение 4. Листинг Tram.cpp 28](#_Toc136866614)

[Приложение 5. Листинг Storage.cpp 28](#_Toc136866615)

[Приложение 6. Листинг Company.hpp 29](#_Toc136866616)

[Приложение 7. Листинг Program.hpp 30](#_Toc136866617)

[Приложение 8. Листинг Storage.hpp 30](#_Toc136866618)

[Приложение 9. Листинг Tram.hpp 30](#_Toc136866619)

[Приложение 10. Листинг TramPark.hpp 31](#_Toc136866620)

# Глава 1. Постановка задачи

Цель работы: Реализовать библиотеку классов, реализующих контейнеры, инкапсулирующие комбинированную структуру данных типа «Динамический неупорядоченный двунаправленный список статических очередей» с информационным наполнением «Горэлектротранс – композиция трампарков(номер), Трампарк – композиция трамваев (марка, регистрационный номер)».

Требования к разрабатываемой программе:

1. Реализация всех необходимых операций (добавление и удаление в основной и присоединённой структурах, поиск в списке).
2. Возможность сохранения всей структуры во внешнем файле (текстовом или XML) с обратной загрузкой.
3. Реализация структуры для хранения и обработки данных конкретной информационной задачи.
4. Именование типов, структур и их полей, классов и их свойств и методов в соответствии с конкретной информационной задачей.
5. Наличие демо-модуля с удобным оконным пользовательским интерфейсом.

Требования к программной документации:

1. Описание всех функций каждого модуля (для чего они предназначены, их входные и выходные параметры, тип и назначение).
2. Описание возможностей, предоставляемых данным программным комплексом (интерфейсная часть).

# Глава 2. Теоретическое описание разработанной структуры

## 2.1. Логическая составляющая структуры

Схематически комбинированная структура «двунаправленный список очередей» выглядит следующим образом:

inf

inf

inf

inf

inf

inf

inf

inf

inf

inf

inf

inf

Главный список

Подчинённые очереди

## 2.2. Набор основных операций

Все операции комбинированной структуры делятся на 3 типа: операции над главной структурой, операции над подчинённой структурой и обход всей структуры.

Операции над главным списком :

1. Инициализация
2. Добавление нового элемента после заданного или перед заданным элементом с проверкой возможности добавления элемента
3. Удаление заданного элемента с проверкой возможности удаления элемента
4. Обход списка
5. Поиск заданного элемента

Операции над подчинённой очередью:

1. Инициализация
2. Добавление элемента с проверкой возможности добавления
3. Удаление элемента с проверкой возможности удаления
4. Обход очереди
5. Поиск заданного элемента

## 2.3. Алгоритмизация основных операций

Операции над главным списком:

* 1. Инициализация
  2. Обход главного списка
* Задание начального значения
* В цикле, пока в списке остались необработанные элементы
* Обработка текущего элемента
* Переход к следующему элементу
  1. Поиск элемента в главном списке
* Проверка возможности поиска
* Запрос информационной части искомого элемента
* Задание начального значения
* В цикле, пока в списке остались необработанные элементы И нужный элемент не найден
* Если информационная часть текущего элемента удовлетворяет заданному критерию, то определить текущий элемент как искомый (найденный) и выйти из цикла, вернув указатель на найденный элемент
* Иначе – переход к следующему элементу
* При выходе из цикла сообщить о том, что поиск был неудачен (передать пустой указатель)
  1. Добавление элемента

1. Проверка возможности добавления
2. Определение положения добавляемого элемента (задание последующего и его поиск)
3. Запрос информационной части нового элемента
4. Определение (инициализация полей) нового элемента
   1. Удаление элемента очереди

* Проверка возможности удаления
* Задание удаляемого элемента (его информационной части)
* Поиск удаляемого элемента
* Обработка удаляемого элемента

Операции над подчинённой структурой:

Отличие алгоритмизации заключается в том, что в начале каждой операции 1-м шагом осуществляется поиск подчинённой очереди в главном списке, а затем уже все остальные шаги.

1. Проверка возможности добавления

* Проверка есть ли свободное место в массиве и добавление элемента

1. Проверка возможности удаления

* Проверка есть ли данный элемент в массиве и удаление элемента

Остальные операции работают так же.

Обход всей структуры:

Обход всей структуры включает в себя:

* Обход главного списка
* При обработке элементов главного списка осуществлять обход соответствующих подчинённых структур

## 2.4. Реализация структуры

Согласно заданию был выбран динамический метод реализации для главной и статический для подчинённой структур.

В соответствии с заданным типом реализации для хранения данных подчинённой структуры будет использоваться массив задаваемого размера.

Для реализации структуры потребуем следующее:

*class Company*

*private:*

*std::string name;*

*TramPark \*pHead;*

*class TramPark*

*private:*

*int number;*

*TramPark \*left;*

*TramPark \*right;*

*Tram array[MAX\_SIZE];*

*int top;*

*int tail;*

*int count;*

Операции над главным списком:

1. Инициализация списка
2. Обход списка

while (tmp != pHead) {

If (tmp->getNumber() == number)

return (tmp);

tmp = (mode == FORWARD ? tmp->getRight() : tmp->getLeft());

}

1. Поиск элемента главного списка

comp->findTramPark(mode, number)

1. Поиск подчиненного элемента

comp->findTramPark(mode, number)->findTram(brand, number)

1. Добавление элемента главного списка

comp->addTramPark(number)

1. Добавление подчиненной очереди

tramParkToFind->addTram(mark, number1)

1. Удаление подчиненной очереди

tramParkToFind->delTram();

1. Удаление элемента главного списка

comp->delTramPark(mode, number)

Операции над подчинённой очередью:

1. Инициализация
2. Добавление элемента

if (count == MAX\_SIZE) {

return false;

}

array[tail] = Tram(brand, year);

tail = (tail + 1) % MAX\_SIZE;

count++;

return true;

1. Удаление элемента

if (count == 0) {

return false;

}

top = (top + 1) % MAX\_SIZE;

count--;

return true;

1. Поиск заданного элемента

int i = top;

do {

if (array[i].getBrand() == brand && array[i].getYear() == year) {

return &array[i];

}

i = (i + 1) % MAX\_SIZE;

} while (i != tail);

return nullptr;

Обход всей структуры:

if (comp->findTramPark(mode, number)) {

if (comp->findTramPark(mode, number)->findTram(brand, number))

}

# Глава 3. Описание программного комплекса

## 3.1. Описание структуры проекта

Для создания проекта была использована среда разработки Microsoft Visual Studio 2022. Проект имеет следующую иерархию:

Kurs93.14

Debug

Kurs93.14.vcxproj

Kurs93.14.sln

… \*

Storage.cpp

Storage.hpp

TramPark.cpp

TramPark.hpp

Tram.cpp

Tram.hpp

Company.cpp

Company.hpp

Kurs.cpp

Program.hpp

… \*\*

Kurs93.14– основная папка проекта.

где Debug – папка, содержащая .exe-файл Debug-версии разработанного консольного приложения, а также дополнительные файлы, необходимые для его работы;

Kurs93.14.vcxproj – файл, хранящий данные текущего проекта;

Kurs93.14.sln – файл решения текущего проекта;

\* – дополнительные файлы, способствующие правильному функционированию проекта в заданной среде разработки (.vs, Kurs93.14.vcxproj.filters, Kurs93.14.vcxproj.user и другие);

Kurs.cpp – файл, содержащий исходный код демонстрационного модуля проекта и точку входа;

\*.h – файлы-заголовков, содержащие описание созданных классов без реализации их методов;

\*.cpp – файлы, содержащий реализацию методов классов, описанных в файле-заголовке \*.h;

\*\* – файлы, создаваемые пользователем для хранения данных о комбинированной структуре (названия задаются пользователями).

## 3.2. Руководство программиста

### 3.2.1. Описание разработанных классов

Для реализации комбинированной структуры было разработано 3 класса: класс Company, реализующий поведение главной структуры

«Горэлектротранс», класс TramPark, реализующий поведение подчинённой структуры «Трампарк», и класс Tram, реализующий поведение элемента нижнего уровня «Трамвай».

* + - 1. **Описание класса Company**

Поля:

TramPark \*phead – указатель на заголовок списка,

std::string name – название компании.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| Company(); | Создаёт заголовок списка, устанавливает значение поля name стандартным значением. |
| Company(std::string name); | Создаёт заголовок списка, устанавливает значение поля name переданным значением. |
| ~Company(); | Производит удаление структуры с очисткой памяти. |
| bool isTramParkListEmpty(); | Проверяет, пуст ли список с депо. |
| bool isTramParkPresent(int mode, int number); | Проверяет наличие трампарка |
| bool addTramPark(int number); | Добавляет трампарк. |
| bool delTramPark(int mode, int number); | Удаляет трампарк. |
| bool addTramParkAfter(int mode, int target, int number); | Добавляет трампарк после. |
| bool addTramParkBefore(int mode, int target, int number); | Добавляет трампарк до. |
| TramPark \*findTramPark(int mode, int number); | Поиск трампарка. |
| void show(); | Вывод на экран элементов структуры. |
| TramPark \*getPHead(); | Возвращает указатель на голову. |
| void setPHead(TramPark \*pHead) | Устанавливает указатель на голову. |
| std::string getName(); | Возвращает имя компании. |
| void setName(std::string name); | Устанавливает имя компании. |
| std::string getString(int mode); | Подготовка строки для вывода в файл. |

* + - 1. **Описание класса TramPark**

Поля:

int top – указатель на начало очереди,

int tail – указатель на конец очереди,

int count – счетчик элементов в очереди,

TramPark \*right – указатель на правый элемент,

TramPark \*left – указатель на левый элемент,

int number – номер трампарка,

Tram array[MAX\_SIZE] – массив трамваев.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| TramPark (int number); | Устанавливает значение поля number переданным значением. |
| bool isEmpty() | Проверяет, пуст ли список с трампарков. |
| bool isFull(); | Проверяет на заполненность. |
| bool addTram(std::string brand, int year); | Добавляет трамвай с маркой и годом. |
| bool delTram(); | Удаляет трамвай. |
| Tram \*findTram(std::string brand, int year); | Поиск трамвая. |
| void show(int mode); | Отображает содержимое. |
| int getNumber(); | Возвращает номер. |
| void setNumber(int number); | Устанавливает номер переданным значением. |
| TramPark \*getRight(); | Возвращает указатель на право. |
| void setRight(TramPark \*right); | Устанавливает указатель на право переданным значением. |
| TramPark \*getLeft(); | Возвращает указатель на лево. |
| void setLeft(TramPark \*left); | Устанавливает указатель на лево переданным значением. |
| std::string getString(); | Подготовка строки для записи в файл. |

**3.2.1.3. Описание класса Tram**

Поля:

std::string brand – марка,

int year – год.

|  |  |
| --- | --- |
| Метод класса | Описание метода |
| Tram(); | Создаёт трамвай с стандартными значениями полей. |
| Tram(std::string brand, int year); | Cоздаёт трамвай с задаными параметрами |
| std::string getBrand(); | Возвращает название бренда. |
| void setBrand(std::string brand); | Устанавливает название бренда. |
| int getYear(); | Возвращает год |
| void setYear(int year); | Устанавливает год |

### 3.2.2. Описание структуры внешнего файла

**3.2.2.1. Информация о записи и загрузке данных с использованием внешнего файла**

Для хранения данных о комбинированной структуре во внешних файлах и последующей загрузки этих данных в основную программу будет использоваться обычный текстовый файл. Сам файл имеет следующую структуру:

* + - * 1. На первой строке в файле находится название компании
        2. Вторая и последующие строки имеют вид:
* Первая колонка – номер трампарка,
* Вторая и последующие чётные колонки (2,4,6,8,10, …, 2\*n) – год,
* Третья и последующие нечётные колонки (3,5,7,9,11, …, 2\*n+1) – марка,

Таким образом, при записи в файл :

«M

3 2010 F5

1 2008 B 2009 A

4 2004 60 2015 AB

2 2005 58 2008 59»

В М в депо под номером 1 окажется два трамвая с годом выпуска 2008 и 2009 марки B,A, соответственно.

**3.2.2.2. Общий вид структуры внешнего файла**

С учётом всех вышеописанных правил, структура файла имеет следующий вид:

<Имя Компании>

<1Трампарк><Год 1><Марка 1><Год 2> <Марка 2>…<Год 2\*n><Марка 2\*n +1>

<2Трампарк><Год 1><Марка 1><Год 2> <Марка 2>…<Год 2\*n><Марка 2\*n +1>

## 3.3. Руководство пользователя

### 3.3.1. Описание демонстрационного модуля

**3.1.1.1. Описание работы с демонстрационным модулем**

После запуска приложения откроется окно консоли, представляющее собой демонстрационный модуль для работы с разработанными классами.

Взаимодействие с пользователем посредством консоли осуществляется следующим образом: пользователю предлагается меню – пронумерованный список команд – из которого требуется выбрать одну команду и указать её номер, а далее, следуя требованиям, указать дополнительную информацию, требуемую для выполнения той или иной программы.

При вводе пользователем тех или иных данных программа проверяет их соответствие требуемому типу данных и адекватность. Если данные корректные, программа продолжает выполнение – в противном случае пользователю сообщается о некорректности введённых данных, после чего предоставляется ещё одна попытка ввода. Программа даёт бесконечное количество попыток ввода до тех пор, пока ввод не окажется корректен.

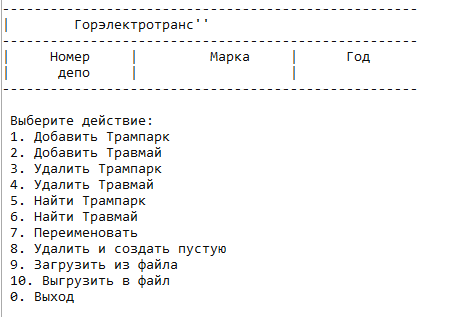
**3.1.1.2. Меню создания Программы**

В момент первого старта программы структура ещё не создана, и пользователь имеет выбор какой обход списка выбрать: прямой или обратный (см. рис. 1).



*Рис. 1. Выбор обхода списка*

Далее пользователю представлена сама структура и меню возможных действий (см. рис. 2).



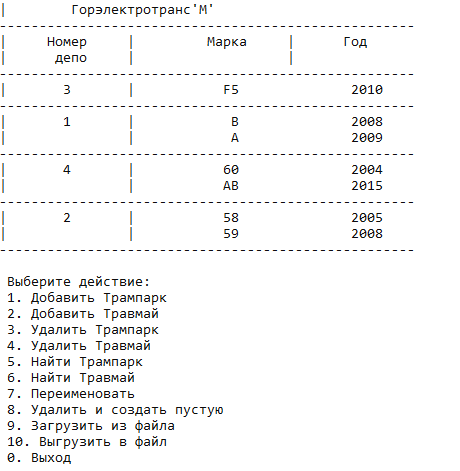
*Рис. 2. Основное меню*

Если пользователь решит загрузить данные из файла и выгрузка будет успешна, программа выведет что выгрузка произошла и предложит пользователю выбрать в каком направлении вывести данные на экран (см. рис. 3).



*Рис. 3. Успешная загрузка данных*

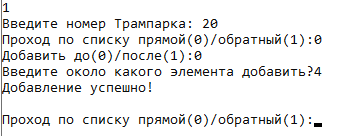
После на экране появятся все данные из файла, а ниже меню возможных действий (см. рис.4):



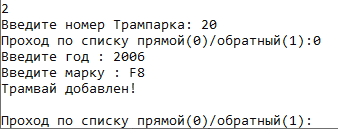
*Рис. 4.*

При добавлении Трампарка пользователь должен будет ввести его номер и выбрать прямой или обратный вывод сделать на экран (см. рис.5).

То же самое при добавлении трамвая, пользователь вводит номер трампарка, год и марку (см. рис.6).

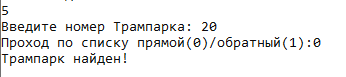


*Рис. 5. Добавление Трампарка*

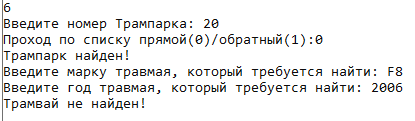


*Рис. 6. Добавление Трамвая*

При поиске Трампарка/трамвая пользователь вводит номер трампарка (или номер трампарка и номер трамвая) (см. рис.7 и рис.8).

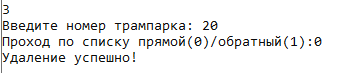


*Рис. 7. Поиск Трампарка*



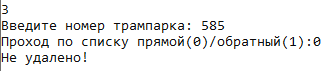
*Рис. 8. Поиск Трамвая*

При удалении пользователь вводит номер Трампарка (Номер Трампарка и номер трамвая) (см. рис.9).



*Рис. 9. Удаление Трампарка*

Если любая из выше перечисленных операций была неудачная пользователь получит подобное сообщение(см. рис.10).



*Рис. 10. Неудачное удаление депо*

Выгрузка в файл при вводе 10 (см. рис.11).



*Рис. 11. Выгрузка в файл*

При некорректном вводе программа попросит повторить ввод(см. рис.12).



*Рис. 12. Некоректнный ввод*

# Заключение

В результате выполнения курсовой работы была реализована заданная комбинированная структура «Динамический неупорядоченный двунаправленный список статических очередей » с информационным наполнением вида Горэлектротранс - композиция трампарков (номер) Трампарк - композиция трамваев (марка, год выпуска) в виде библиотеки разработанных классов. Был создан пользовательский интерфейс, демонстрирующий работу с заданной структурой . Кроме этого, были разработаны руководство разработчика и руководство пользователя по работе с данным приложением.

# Литература

1. *Козин А. Н.* Структуры и алгоритмы обработки данных. Учебно-методическое пособие. Издательство: Татарский институт содействия бизнесу, 2003.
2. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений. Москва: Бином, 2000.
3. *Вирт Н.* Алгоритмы и структуры данных. Москва: Мир, 1989.

# Приложения

## Приложение 1. Листинг Kurs.cpp

#include "Company.hpp"

#include "Program.hpp"

#include "Storage.hpp"

#include "Tram.hpp"

#include "TramPark.hpp"

#include <Windows.h>

Company \*comp = NULL;

Storage file("text.txt");

int getNumber() {

int number;

std::cin >> number;

while (!std::cin || number < 0) {

std::cout << "Введено некорректное число, повторите: ";

std::cin.clear();

std::cin.ignore(100500, '\n');

std::cin >> number;

}

std::cin.ignore(100500, '\n');

return (number);

}

int getMode() {

int number;

std::cin >> number;

while (!std::cin || (number != 0 && number != 1)) {

std::cout << "Введено некорректное число, повторите: ";

std::cin.clear();

std::cin.ignore(100500, '\n');

std::cin >> number;

}

std::cin.ignore(100500, '\n');

return (number);

}

void Add() {

int number, mode = 0;

std::cout << "Введите номер Трампаркa: ";

number = getNumber();

if (!comp->isTramParkListEmpty()) {

std::cout << "Проход по списку прямой(0)/обратный(1):";

mode = getMode();

}

if (comp->isTramParkListEmpty()) {

if (comp->addTramPark(number))

std::cout << "Добавление успешно!\n";

} else if (!comp->findTramPark(mode, number)) {

std::cout << "Добавить до(0)/после(1):";

int pos = getMode();

std::cout << "Введите около какого элемента добавить?";

int target = getNumber();

if (pos == 1 ? comp->addTramParkAfter(mode, target, number)

: comp->addTramParkBefore(mode, target, number)) {

std::cout << "Добавление успешно!\n";

} else

std::cout << "не добавлено!\n";

} else

std::cout << "Такой трампарк уже есть в списке!";

}

void AddL() {

int number, number1, mode;

std::cout << "Введите номер Трампарка: ";

number = getNumber();

std::cout << "Проход по списку прямой(0)/обратный(1):";

mode = getMode();

TramPark \*tramParkToFind = comp->findTramPark(mode, number);

if (tramParkToFind != nullptr) {

if (tramParkToFind->isFull()) {

std::cout << "Достигнут лимит массива" << std::endl;

} else {

std::cout << "Введите год : ";

number1 = getNumber();

std::string mark;

std::cout << "Введите марку : ";

std::getline(std::cin, mark);

if (mark.length() == 0) {

std::cout << "Плохие данные" << std::endl;

} else {

if (tramParkToFind->findTram(mark, number1) == nullptr) {

bool res = tramParkToFind->addTram(mark, number1);

if (res) {

std::cout << "Трамвай добавлен!\n";

} else {

std::cout << "Трамвай не добавлен!\n";

}

} else

std::cout << "Такой трамвай уже есть в списке!\n";

}

}

} else

std::cout << "Такого отдела нет!\n";

}

void Del() {

int number, mode;

std::cout << "Введите номер трампарка: ";

number = getNumber();

std::cout << "Проход по списку прямой(0)/обратный(1):";

mode = getMode();

if (comp->delTramPark(mode, number))

std::cout << "Удаление успешно!\n";

else

std::cout << "Не удалено!\n";

}

void DelL() {

int number, numberL, mode;

std::string brand;

std::cout << "Введите номер Трампарка, в котором требуется удалить Трамвай ";

number = getNumber();

std::cout << "Проход по списку прямой(0)/обратный(1):";

mode = getMode();

TramPark \*tramParkToFind = comp->findTramPark(mode, number);

if (tramParkToFind != nullptr) {

if (!tramParkToFind->isEmpty()) {

std::cout << "Удаление успешно!\n";

tramParkToFind->delTram();

} else {

std::cout << "Нет записей.\n";

}

} else

std::cout << "Такого трампарка нету!\n";

}

void Find() {

int number, mode;

std::cout << "Введите номер Трампарка: ";

number = getNumber();

std::cout << "Проход по списку прямой(0)/обратный(1):";

mode = getMode();

if (comp->findTramPark(mode, number))

std::cout << "Трампарк найден!\n";

else

std::cout << "Трампарк не найден!";

}

void FindL() {

int number, mode, number1;

std::cout << "Введите номер Трампарка: ";

number = getNumber();

std::cout << "Проход по списку прямой(0)/обратный(1):";

mode = getMode();

if (comp->findTramPark(mode, number)) {

std::string brand;

std::cout << "Трампарк найден!\n";

std::cout << "Введите марку травмая, который требуется найти: ";

std::getline(std::cin, brand);

std::cout << "Введите год травмая, который требуется найти: ";

int numberL = getNumber();

if (comp->findTramPark(mode, number)->findTram(brand, number)) {

std::cout << "Трамвай найден!" << std::endl;

} else {

std::cout << "Трамвай не найден!" << std::endl;

}

} else

std::cout << "Трампарк не найден!" << std::endl;

;

}

void Show() {

{

int mode;

std::cout << "\nПроход по списку прямой(0)/обратный(1):";

mode = getMode();

system("cls");

std::cout << "\n----------------------------------------------------\n";

std::cout << "| Горэлектротранс'" << comp->getName() << "'";

std::cout << "\n----------------------------------------------------";

std::cout << "\n| Номер |"

<< " Марка | Год ";

std::cout << "\n| депо |"

<< " |\n";

comp->show(mode);

std::cout << "----------------------------------------------------\n";

}

}

int main() {

SetConsoleCP(65001);

SetConsoleOutputCP(65001);

int n;

comp = new Company();

while (true) {

Show();

std::cout << "\n Выберите действие:\n";

std::cout << " 1. Добавить Трампарк\n";

std::cout << " 2. Добавить Травмай\n";

std::cout << " 3. Удалить Трампарк\n";

std::cout << " 4. Удалить Травмай\n";

std::cout << " 5. Найти Трампарк\n";

std::cout << " 6. Найти Травмай\n";

std::cout << " 7. Переименовать\n";

std::cout << " 8. Удалить и создать пустую\n";

std::cout << " 9. Загрузить из файла\n";

std::cout << " 10. Выгрузить в файл\n";

std::cout << " 0. Выход\n";

n = getNumber();

if (n == 1)

Add();

else if (n == 2)

AddL();

else if (n == 3)

Del();

else if (n == 4)

DelL();

else if (n == 5)

Find();

else if (n == 6)

FindL();

else if (n == 7) {

std::string newName;

std::cout << "Введите название компании: ";

std::cin >> newName;

comp->setName(newName);

} else if (n == 8) {

delete (comp);

comp = new Company();

} else if (n == 9) {

delete (comp);

comp = new Company();

if (file.InputData(comp))

std::cout << "Загрузка успешна!";

else {

std::cout << "Ошибка!";

delete (comp);

comp = new Company();

}

} else if (n == 10) {

if (comp != NULL) {

if (file.OutputData(comp, 0))

std::cout << "Выгрузка успешна!";

else

std::cout << "Ошибка";

} else

std::cout << "Ошибка!";

} else if (n == 0) {

delete (comp);

break;

}

}

}

## Приложение 2. Листинг Company.cpp

#include "Company.hpp"

#include "Program.hpp"

#include "TramPark.hpp"

Company::Company() {

pHead = new TramPark(-1);

pHead->setRight(pHead);

pHead->setLeft(pHead);

name = "";

}

Company::Company(std::string name) {

pHead = new TramPark(-1);

pHead->setRight(pHead);

pHead->setLeft(pHead);

this->name = name;

}

Company::~Company() {

TramPark \*pPrev;

TramPark \*pCurrent;

pCurrent = pHead->getRight();

while (pCurrent != pHead) {

pPrev = pCurrent;

pCurrent = pCurrent->getRight();

delete pPrev;

}

delete pHead;

}

bool Company::isTramParkListEmpty() {

return (pHead->getRight() == pHead && pHead->getLeft() == pHead);

}

bool Company::isTramParkPresent(int mode, int number) {

TramPark \*tmp = (mode == FORWARD ? pHead->getRight() : pHead->getLeft());

while (tmp != pHead) {

if (tmp->getNumber() == number)

return (true);

tmp = (mode == FORWARD ? tmp->getRight() : tmp->getLeft());

}

return (false);

}

TramPark \*Company::findTramPark(int mode, int number) {

TramPark \*tmp = (mode == FORWARD ? pHead->getRight() : pHead->getLeft());

while (tmp != pHead) {

if (tmp->getNumber() == number)

return (tmp);

tmp = (mode == FORWARD ? tmp->getRight() : tmp->getLeft());

}

return (nullptr);

}

bool Company::addTramPark(int number) {

TramPark \*pNew = new TramPark(number);

pNew->setRight(pHead);

pNew->setLeft(pHead->getLeft());

pHead->getLeft()->setRight(pNew);

pHead->setLeft(pNew);

return (true);

}

bool Company::addTramParkBefore(int mode, int target, int number) {

TramPark \*pCurrent = (mode == FORWARD ? pHead->getRight() : pHead->getLeft());

while (pCurrent != pHead) {

if (pCurrent->getNumber() == target) {

TramPark \*pNew = new TramPark(number);

pNew->setLeft(pCurrent->getLeft());

pNew->setRight(pCurrent);

pCurrent->getLeft()->setRight(pNew);

pCurrent->setLeft(pNew);

return (true);

}

pCurrent = (mode == FORWARD ? pCurrent->getRight() : pCurrent->getLeft());

}

return (false);

}

bool Company::addTramParkAfter(int mode, int target, int number) {

TramPark \*pCurrent = (mode == FORWARD ? pHead->getRight() : pHead->getLeft());

while (pCurrent != pHead) {

if (pCurrent->getNumber() == target) {

TramPark \*pNew = new TramPark(number);

pNew->setRight(pCurrent->getRight());

pNew->setLeft(pCurrent);

pCurrent->getRight()->setLeft(pNew);

pCurrent->setRight(pNew);

return (true);

}

pCurrent = (mode == FORWARD ? pCurrent->getRight() : pCurrent->getLeft());

}

return (false);

}

bool Company::delTramPark(int mode, int number) {

TramPark \*pCurrent = (mode == FORWARD ? pHead->getRight() : pHead->getLeft());

while (pCurrent != pHead) {

if (pCurrent->getNumber() == number) {

pCurrent->getLeft()->setRight(pCurrent->getRight());

pCurrent->getRight()->setLeft(pCurrent->getLeft());

delete pCurrent;

return (true);

}

pCurrent = (mode == FORWARD ? pCurrent->getRight() : pCurrent->getLeft());

}

return (false);

}

void Company::show(int mode) {

if (!isTramParkListEmpty()) {

TramPark \*tmp = (mode == FORWARD ? pHead->getRight() : pHead->getLeft());

while (tmp != pHead) {

std::cout << "----------------------------------------------------\n";

std::cout << "|" << std::setw(8) << tmp->getNumber() << std::setw(8)

<< "|";

tmp->show(0);

std::cout << "\n";

tmp = (mode == FORWARD ? tmp->getRight() : tmp->getLeft());

}

}

}

std::string Company::getString(int mode) {

std::string str = "";

TramPark \*tmp = (mode == FORWARD ? pHead->getRight() : pHead->getLeft());

while (tmp != pHead) {

std::stringstream ss;

ss << tmp->getNumber();

str += ss.str();

str += tmp->getString();

str += '\n';

tmp = mode == FORWARD ? tmp->getRight() : tmp->getLeft();

}

return str;

}

std::string Company::getName() { return name; }

void Company::setName(std::string name) { this->name = name; }

TramPark \*Company::getPHead() { return pHead; }

void Company::setPHead(TramPark \*pHead) { this->pHead = pHead; }

## Приложение 3. Листинг Trampark.cpp

#include "TramPark.hpp"

#include "Program.hpp"

TramPark::TramPark(int number) {

this->number = number;

left = nullptr;

right = nullptr;

top = 0;

tail = 0;

count = 0;

}

bool TramPark::isEmpty() {

return count == 0;

}

Tram\* TramPark::findTram(std::string brand, int year) {

int i = top;

do {

if (array[i].getBrand() == brand && array[i].getYear() == year) {

return &array[i];

}

i = (i + 1) % MAX\_SIZE;

} while (i != tail);

return nullptr;

}

bool TramPark::addTram(std::string brand, int year) {

if (count == MAX\_SIZE) {

return false;

}

array[tail] = Tram(brand, year);

tail = (tail + 1) % MAX\_SIZE;

count++;

return true;

}

bool TramPark::delTram() {

if (count == 0) {

return false;

}

top = (top + 1) % MAX\_SIZE;

count--;

return true;

}

void TramPark::show(int mode) {

int i = top;

do {

if (array[i].getBrand() != "") {

Tram tmp = array[i];

if (i != top)

std::cout << "|" << std::setw(16) << "|";

std::cout << std::setw(13) << tmp.getBrand() << std::setw(8) << " "

<< std::setw(10) << tmp.getYear();

if ((i + 1) % MAX\_SIZE != tail)

std::cout << std::endl;

}

i = (i + 1) % MAX\_SIZE;

} while (i != tail);

}

std::string TramPark::getString() {

int temp;

std::string str = "";

for (int i = top; i != tail; i = (i + 1) % MAX\_SIZE) {

if (array[i].getBrand() != "") {

std::stringstream ss;

temp = array[i].getYear();

str += " ";

ss << temp;

str += ss.str();

str += " ";

str += array[i].getBrand();

}

}

return str;

}

bool TramPark::isFull() {

return count == MAX\_SIZE;

}

int TramPark::getNumber() {

return number;

}

void TramPark::setNumber(int number) {

this->number = number;

}

TramPark\* TramPark::getLeft() {

return left;

}

void TramPark::setLeft(TramPark\* left) {

this->left = left;

}

TramPark\* TramPark::getRight() {

return right;

}

void TramPark::setRight(TramPark\* right) {

this->right = right;

}

## Приложение 4. Листинг Tram.cpp

#include "Tram.hpp"

Tram::Tram() {

this->brand = "";

this->year = 0;

}

Tram::Tram(std::string brand, int year) {

this->brand = brand;

this->year = year;

}

std::string Tram::getBrand() { return brand; }

void Tram::setBrand(const std::string brand) { this->brand = brand; }

int Tram::getYear() { return year; }

void Tram::setYear(int year) { this->year = year; }

## Приложение 5. Листинг Storage.cpp

#include "Storage.hpp"

#include "Program.hpp"

Storage::Storage(std::string address) { this->address = address; };

bool Storage::InputData(Company \*comp) {

std::string name, fuel;

int number, number1;

char buffer[100];

char \*f;

std::ifstream file(address);

if (!file.is\_open())

return false;

file.getline(buffer, 100);

if (buffer != " ") {

name = buffer;

comp->setName(name);

} else {

std::cout << "error" << std::endl;

return false;

}

while (!file.eof()) {

file.getline(buffer, 100);

f = strtok(buffer, " ");

if (f == NULL)

continue;

number = atoi(f);

comp->addTramPark(number);

while (f) {

f = strtok(NULL, " ");

if (!f) {

break;

}

number1 = atoi(f);

f = strtok(NULL, " ");

fuel = f;

comp->findTramPark(0, number)->addTram(fuel, number1);

}

}

file.close();

return true;

}

bool Storage::OutputData(Company \*comp, int mode) {

std::ofstream file(address);

if (!file.is\_open())

return false;

file << comp->getName() + '\n' + comp->getString(mode);

file.close();

return true;

}

Storage::~Storage(){};

## Приложение 6. Листинг Company.hpp

#ifndef COMPANY\_HPP\_

#define COMPANY\_HPP\_

#include "Tram.hpp"

#include "TramPark.hpp"

#include <iostream>

class Company {

private:

std::string name;

TramPark \*pHead;

public:

Company();

Company(std::string name);

~Company();

bool isTramParkListEmpty();

bool isTramParkPresent(int mode, int number);

bool addTramPark(int number);

bool addTramParkAfter(int mode, int target, int number);

bool addTramParkBefore(int mode, int target, int number);

bool delTramPark(int mode, int number);

TramPark \*findTramPark(int mode, int number);

void show(int mode);

std::string getString(int mode);

std::string getName();

void setName(std::string name);

TramPark \*getPHead();

void setPHead(TramPark \*pHead);

};

#endif

## Приложение 7. Листинг Program.hpp

#ifndef PROGRAM\_HPP

#define PROGRAM\_HPP

#include <cstring>

#include <fstream>

#include <iomanip>

#include <iostream>

#include <sstream>

#include <string.h>

#include <stdlib.h>

#define FORWARD 0

#define BACKWARD 1

#endif

## Приложение 8. Листинг Storage.hpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#ifndef STORAGE\_HPP

#define STORAGE\_HPP

#include "Company.hpp"

#include "Program.hpp"

class Storage {

private:

std::string address;

public:

Storage(std::string address);

bool InputData(Company \*\_comp);

bool OutputData(Company \*\_comp, int mode);

~Storage();

};

#endif

## Приложение 9. Листинг Tram.hpp

#ifndef TRAM\_HPP\_

#define TRAM\_HPP\_

#include <iostream>

#include <string>

class Tram {

private:

std::string brand;

int year;

public:

Tram();

Tram(std::string brand, int year);

std::string getBrand();

void setBrand(std::string brand);

int getYear();

void setYear(int year);

};

#endif

## Приложение 10. Листинг TramPark.hpp

#ifndef TRAMPARK\_HPP\_

#define TRAMPARK\_HPP\_

#include "Tram.hpp"

const int MAX\_SIZE = 4;

class TramPark {

private:

int number;

TramPark \*left;

TramPark \*right;

Tram array[MAX\_SIZE];

int top;

int tail;

int count;

public:

TramPark(int number);

bool isEmpty();

Tram \*findTram(std::string brand, int year);

bool addTram(std::string brand, int year);

bool delTram();

bool isFull();

void show(int mode);

std::string getString();

int getNumber();

void setNumber(int number);

TramPark \*getLeft();

void setLeft(TramPark \*left);

TramPark \*getRight();

void setRight(TramPark \*right);

};

#endif