ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ПРАКТИЧЕСКОЙ РАБОТЕ

По дисциплине «Языки программирования»

Выполнил: ст. гр. ТКИ – 241

Владимиров М. А.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2024

Содержание

1. Формулировка задачи

2. UML диаграмма

3. Решение задачи на языке программирования C++

4. Тесты

5. Зачет задания в GitHub

**Задание 4–1**

1. **Формулировка задачи**

Написать программу на языке С++ которая реализует линейный односвязный список с использованием умных указателей.

1. **UML диаграмма**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

1. **Решение задачи на языке программирования C++**

﻿**List.h**

﻿#pragma once

#include <iostream>

#include <exception>

#include <string>

#include <sstream>

#include <memory>

template<typename T>

class List

{

public:

/\*\*

@brief конструктор класса List.

\*/

List();

/\*\*

\* @brief Деструктор

\*/

~List();

/\*\*

@brief Функция добавляющая элемент в конец списка.

@param data Данные добавляемые в список.

\*/

void PushBack(const T& data);

/\*

@brief Функция создающая строку из списка

\*/

std::string ToString() const noexcept;

/\*\*

@brief Функция удаляющая элемент с начала списка.

\*/

void PopFront();

/\*\*

@brief Функция добавляющая элемент в начало списка.

@param data Данные добавляемые в список.

\*/

void PushFront(const T& data);

/\*\*

@brief Конструктор списка через initializer\_list.

@param list Список.

\*/

List(std::initializer\_list<T> list);

/\*\*

@brief Функция очистки списка

\*/

void Clear();

/\*\*

\* @brief Перегрузка оператора для копирования.

\* @param other Список для копирования.

\* @return Скопированный список.

\*/

List<T>& operator=(const List<T>& other);

/\*\*

\* @brief Перегрузка оператора для мува.

\* @param other Список для мува.

\* @return Мувнутый список.

\*/

List<T>& operator=(List<T>&& other) = default;

/\*

\* @brief Переопределение оператора сравнения.

\* @param list Список с которым сравниваем.

\* @return this->ToString() == list.ToString().

\*/

bool operator==(List<T>& list) noexcept;

/\*\*

\* @brief Проверка наличия элементов в списке.

\* @return true Если список пустой.

\*/

bool IsEmpty() noexcept;

/\*\*

\* @brief Конструктор копирования

\*/

List<T>(const List<T>& other) = default;

/\*\*

\* @brief Мув

\*/

List<T>(List&& second) = default;

private:

template<typename T>

struct Node

{

T data;

std::unique\_ptr<Node<T>> pNext;

/\*\*

\* @brief Конструктор класса Node.

\* @param data Данные.

\* @param pNext Указатель на следующий элемент списка.

\*/

Node(const T& data, Node<T>\* pNext = nullptr) : data(data), pNext(pNext)

{

};

};

size\_t size;

std::unique\_ptr<Node<T>> head;

};

template<typename T>

inline List<T>::List() : size(0), head(nullptr)

{

}

template<typename T>

inline List<T>::~List()

{

while (head != nullptr)

{

auto pNext = std::move(head->pNext);

head = std::move(pNext);

}

}

template<typename T>

inline void List<T>::PushBack(const T& data)

{

if (head == nullptr)

{

head = std::make\_unique<Node<T>>(data);

}

else

{

auto current = this->head.get();

while (current->pNext != nullptr)

{

current = current->pNext.get();

}

current->pNext = std::make\_unique<Node<T>>(data);

}

size++;

}

template<typename T>

inline std::string List<T>::ToString() const noexcept

{

std::stringstream buffer{};

auto current = this->head.get();

while (current != nullptr)

{

buffer << current->data;

current = current->pNext.get();

}

return buffer.str();

}

template<typename T>

inline void List<T>::PopFront()

{

if (head == nullptr)

{

throw std::invalid\_argument("Список пуст!");

}

else

{

head = std::move(head->pNext);

size--;

}

}

template<typename T>

inline void List<T>::PushFront(const T& data)

{

if (head == nullptr)

{

head = std::make\_unique<Node<T>>(data);

}

else

{

std::unique\_ptr<Node<T>> newNode = std::make\_unique<Node<T>>(data,

head.release());

head = std::move(newNode);

}

size++;

}

template<typename T>

inline List<T>::List(std::initializer\_list<T> list) : List()

{

for (auto& data : list)

{

this->PushBack(data);

}

}

template<typename T>

inline void List<T>::Clear()

{

while (size > 0)

{

PopFront();

}

}

template<typename T>

inline List<T>& List<T>::operator=(const List& other)

{

if (this != &other)

{

this->Clear();

Node<T>\* current = other.head.get();

while (current != nullptr)

{

this->PushBack(current->data);

current = current->pNext.get();

}

}

return \*this;

}

template<typename T>

inline bool List<T>::operator==(List<T>& list) noexcept

{

return this->ToString() == list.ToString();

}

template<typename T>

inline bool List<T>::IsEmpty() noexcept

{

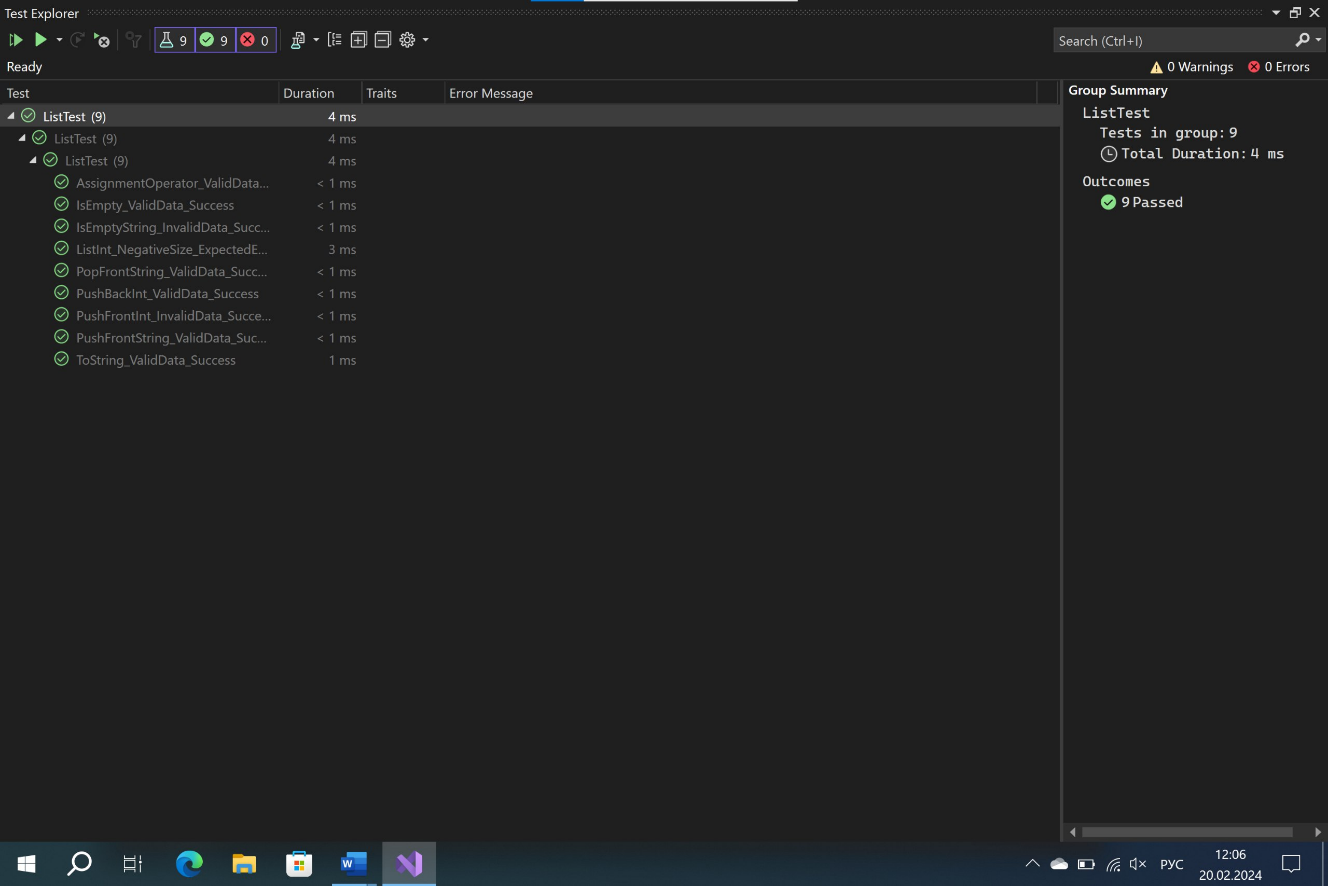
return this->head == nullptr;

}

**List.cpp**

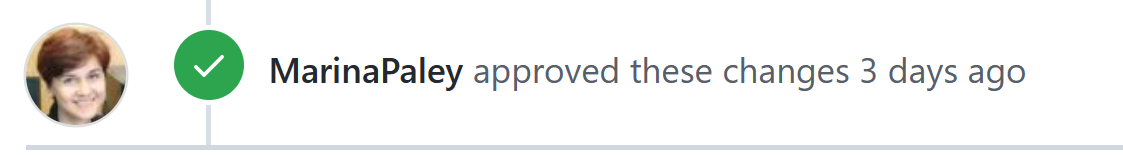
﻿#include "List.h"

1. **Тесты**

****

**Рисунок 1 - тесты**

1. **Зачет задания в GitHub**



**Рисунок 2 – Зачет задания в GitHub**