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 № 1

По дисциплине «Языки программирования»

Выполнил: ст. гр. ТКИ – 241

Дьячков Денис Андреевич

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

**Формулировка задания:**

Написать односвязный линейный список с использованием темплейтов

**Код:**

#pragma once

#include <iostream>

#include <sstream>

namespace miit::list

{

template<typename T>

class Node;

template<typename T>

class List;

/\*\*

\* @brief Перегрузка оператора равно

\* @param rha Первый аргумент для сравнения

\* @param lha Второй аргумент для сравнения

\* @return false если равны и true если н ет

\*/

template<typename T>

bool operator==(const List<T>& rha, const List<T>& lha) noexcept;

/\*\*

\* @brief Перегрузка оператора не равно

\* @param rha Первый аргумент для сравнения

\* @param lha Второй аргумент для сравнения

\* @return true если равны и false если нет

\*/

template<typename T>

bool operator!=(const List<T>& rha, const List<T>& lha) noexcept;

/\*\*

\* @brief Перегрузка оператора вывода

\* @return Поток вывода со списком

\*/

template<typename T>

std::ostream& operator<<(std::ostream& os, List<T>& list) noexcept;

template<typename T>

class List final

{

private:

Node<T>\* head;

Node<T>\* tail;

public:

/\*\*

\* @brief Создает объект типа List

\*/

List();

/\*\*

\* @brief Создает объект типа List по initializer\_list'у

\* @param list Список по которому будет создаваться список

\*/

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

/\*\*

\* @brief Чистит память за объектом типа List

\*/

~List();

/\*\*

\* @brief Перегрузка оператора копирования

\* @param list Список для копирования

\* @return Список скопированный с list

\*/

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

/\*\*

\* @brief Перегрузка оператора перемещения

\* @param list Список для мува

\* @return Список после мува

\*/

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

/\*\*

\* @brief Перегрузка конструктора копирования

\* @param list Список для копирования

\*/

List(const List<T>& list);

/\*\*

\* @brief Перегрузка конструктора перемещения

\* @param list Список для мува

\*/

List(List<T>&& list) noexcept;

/\*\*

\* @brief Удаление всех элементов списка

\*/

void clear();

/\*\*

\* @brief Добавление элемента в начало списка

\* @param value Элемент для добавления

\*/

void push\_forward(T value);

/\*\*

\* @brief Добавление элемента в конец списка

\* @param value Элемент для добавления

\*/

void push\_back(T value);

/\*\*

\* @brief Удаление элемента с начала списка

\*/

void pop\_forward();

/\*\*

\* @brief Есть ли элементы в списке

\* @return true - если список не пустой и false если он пустой

\*/

bool has\_elements() const;

/\*\*

\* @brief Превращение списка в строку

\* @return Строка по списку

\*/

std::string to\_string() const;

};

template<typename T>

class Node

{

public:

/\*\*

\* @brief Создает объект типа Node

\* @param value Значение ноды

\* @param next\_node Следующая нода

\*/

Node(T value, Node<T>\* next\_node);

T value;

Node<T>\* next\_node;

};

template<typename T>

inline Node<T>::Node(T value, Node<T>\* next\_node)

:value(value), next\_node(next\_node)

{

}

template<typename T>

inline List<T>::List()

:head(nullptr), tail(nullptr)

{

}

template<typename T>

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

: head(nullptr), tail(nullptr)

{

for (auto value : list)

{

this->push\_back(value);

}

}

template<typename T>

inline List<T>::~List()

{

this->clear();

}

template<typename T>

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

{

if (\*this != list)

{

List temp{ list };

std::swap(temp.head, this->head);

std::swap(temp.tail, this->tail);

}

return \*this;

}

template<typename T>

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

{

if (\*this != list)

{

\*this = std::move(list);

}

return \*this;

}

template<typename T>

inline List<T>::List(const List<T>& list)

:head(nullptr), tail(nullptr)

{

List temp{};

Node<T>\* curr = list.head;

while (curr != nullptr)

{

temp.push\_back(curr->value);

curr = curr->next\_node;

}

std::swap(temp.head, this->head);

std::swap(temp.tail, this->tail);

}

template<typename T>

inline List<T>::List(List<T>&& list) noexcept

:head(nullptr), tail(nullptr)

{

\*this = std::move(list);

}

template<typename T>

inline void List<T>::clear()

{

while (this->has\_elements())

{

this->pop\_forward();

}

}

template<typename T>

inline void List<T>::push\_forward(T value)

{

Node<T>\* next\_head = new Node<T>(value, this->head);

this->head = next\_head;

if (!this->has\_elements())

{

this->tail = next\_head;

}

}

template<typename T>

inline void List<T>::push\_back(T value)

{

Node<T>\* next\_tail = new Node<T>(value, nullptr);

if (this->has\_elements())

{

this->tail->next\_node = next\_tail;

this->tail = next\_tail;

}

else

{

this->tail = next\_tail;

this->head = next\_tail;

}

}

template<typename T>

inline void List<T>::pop\_forward()

{

if (!this->has\_elements())

{

throw std::out\_of\_range("List is Empty");

}

Node<T>\* next\_head = this->head->next\_node;

delete this->head;

this->head = next\_head;

}

template<typename T>

inline bool List<T>::has\_elements() const

{

return this->head != nullptr;

}

template<typename T>

inline std::string List<T>::to\_string() const

{

std::stringstream buffer;

Node<T>\* curr = this->head;

while (curr != nullptr)

{

buffer << curr->value;

buffer << " ";

curr = curr->next\_node;

}

return buffer.str();

}

template<typename T>

bool operator==(const List<T>& rha, const List<T>& lha) noexcept

{

return (rha.to\_string() == lha.to\_string());

}

template<typename T>

bool operator!=(const List<T>& rha, const List<T>& lha) noexcept

{

return !(rha == lha);

}

template<typename T>

std::ostream& operator<<(std::ostream& os, List<T>& list) noexcept

{

return os << list.to\_string();

}

}

**Код тестов:**

#include "pch.h"

#include "CppUnitTest.h"

#include "../Template\_list/List.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

using namespace miit::list;

namespace Tests

{

TEST\_CLASS(Tests)

{

public:

TEST\_METHOD(TestMethod1)

{

Assert::IsTrue(true);

}

TEST\_METHOD(HasElements\_ValidData\_Success)

{

List list{ 1, 22, 3 ,-1 ,999 };

Assert::IsTrue(list.has\_elements());

}

TEST\_METHOD(ToString\_ValidData\_Success)

{

List list{ 1, 22, 3 ,-1 ,999 };

Assert::IsTrue(list.to\_string() == "1 22 3 -1 999 ");

}

TEST\_METHOD(PushForward\_ValidData\_Success)

{

List list{ 1, 22, 3 ,-1 ,999 };

list.push\_forward(100);

List expected\_lsit{ 100, 1, 22, 3 ,-1 ,999 };

Assert::IsTrue(list.to\_string() == list.to\_string());

}

TEST\_METHOD(PushBack\_ValidData\_Success)

{

List list{ 1, 22, 3 ,-1 ,999 };

list.push\_forward(100);

List expected\_lsit{ 1, 22, 3 ,-1 ,999, 100 };

Assert::IsTrue(list.to\_string() == list.to\_string());

}

TEST\_METHOD(PushBackString\_ValidData\_Success)

{

List list{ "1", "22", "3" ,"-1" ,"999" };

list.push\_forward("100");

List expected\_lsit{ "1", "22", "3" ,"-1" ,"999", "100" };

Assert::IsTrue(list.to\_string() == list.to\_string());

}

TEST\_METHOD(PopForward\_ValidData\_Success)

{

List list{ 1, 22, 3 ,-1 ,999 };

list.pop\_forward();

List expected\_lsit{ 22, 3 ,-1 ,999 };

Assert::IsTrue(list.to\_string() == list.to\_string());

}

TEST\_METHOD(CoppyOperator\_ValidData\_Success)

{

List list\_1{ 1, 22, 3 ,-1 ,999 };

List list\_2{ 3, 2, 555 ,-233 ,17423 };

list\_1 = list\_2;

Assert::IsTrue(list\_1.to\_string() == list\_2.to\_string());

}

TEST\_METHOD(Pop\_AllreadyEmpty\_ExeptionExpected)

{

Assert::ExpectException<std::out\_of\_range>([]()

{List<int> list = List<int>(); list.pop\_forward(); });

}

TEST\_METHOD(OutputOperator\_ValidData\_Success)

{

std::stringstream buffer;

List list\_1{ 1, 22, 3 ,-1 ,999 };

buffer << list\_1;

Assert::IsTrue(buffer.str() == "1 22 3 -1 999 ");

}

};

}

**Прохождение тестов:**

**Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание**

Рисунок 1 - Пройденные тесты

**UML диаграмма кода:**

**Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание**

Рисунок 2 - UML диаграмма кода

**Approve:**

Рисунок 3 - Approve