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 ЛАБОРАТОРНОЙ РАБОТЕ №2

По дисциплине «ООП»

ВАРИАНТ 3

Выполнил: ст. гр. ТКИ-141

Блынский Герман Вадимович

Проверил:

Москва 2024

Оглавление

[1 Решение задачи 2](#_Toc168890080)

[1.2 UML диаграмма алгоритма задание 3](#_Toc168890081)

[1.3 Текст программы на языке С++ 4](#_Toc168890082)

[1.4 Результаты выполнения программы 11](#_Toc168890083)

[1.5 Выполнение тестовых примеров задание 12](#_Toc168890084)

[1.6 Отметка о выполнении задания в веб-хостинге системы контроля версий задание 13](#_Toc168890085)

1. Решение задачи

**1.1 Формулировка** **задачи**

**Вариант 2. Линейный двусвязный список**

Создать решение (Solution), которое минимально содержит три проекта (Projects): исполняемый (Console Application), библиотеку классов (Library), и модульные тесты (Tests). Разработать библиотеку классов по заданному варианту. Важно! Библиотека классов не должна зависеть от потоков ввода-вывода. Каждый класс необходимо размещать в отдельных двух файлах, снабжённых «говорящим именем» и специальными расширениями: \*.h для заголовочных файлов (Header), содержащих API класса, и \*.cpp для компилируемых (Source), содержащих реализацию класса. В запускаемом проекте требуется создать файл main.cpp, содержащий точку входа в демонстрационную программу – главную функцию (main). В рамках данной функции показать работу с коллекцией.

Реализовать библиотеку заданной (по вариантам) структуры данных целых чисел. Предусмотреть конструктор со списком инициализации (https://en.cppreference.com/w/cpp/utility/initializer\_list). Реализовать все конструкторы, создаваемые компилятором по умолчанию, реализовать деструктор. Предусмотреть методы вывода в строку содержимого коллекции. Переопределить операторы присваивания, сдвига влево и сдвига вправо. Предусмотреть метод определения наличия элементов коллекции (пустая ли коллекция?).

1.2 UML диаграмма алгоритма задание

UML диаграмма представлена ниже (**Ошибка! Источник ссылки не найден.**).

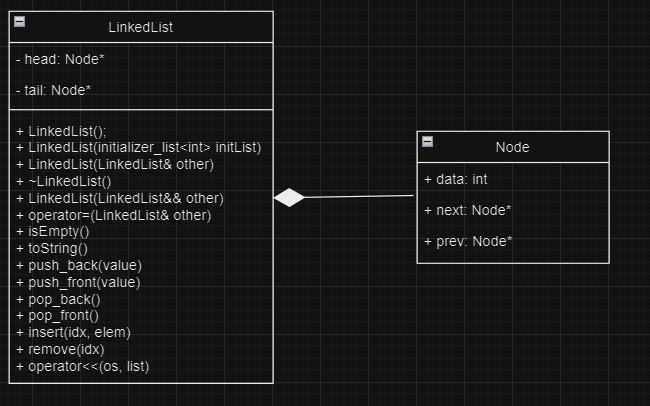


Рисунок 1 - UML диаграмма

1.3 Текст программы на языке С++

//Файл Node.h

#pragma once

struct Node

{

int data;

Node\* next;

Node\* prev;

};

//Файл Node.cpp

#include "Node.h"

//Файл List.h

#include "Node.h"

#include <iostream>

#include <string>

#pragma once

namespace rut\_miit

{

class LinkedList

{

private:

Node\* head;

Node\* tail;

public:

/\*\*

\* @brief - Конструктор по умолчанию

\*/

LinkedList();

/\*\*

\* @brief - Конструктор, создающий список по значениям, заданным пользователем

\*/

LinkedList(std::initializer\_list<int> initList);

/\*\*

\* @brief - Конструктор копирования

\* @param other - Название уже существующего списка

\*/

LinkedList(const LinkedList& other);

/\*\*

\* @brief - Деструктор по умолчанию

\*/

~LinkedList();

/\*\*

\* @brief - Конструктор перемещения

\*/

LinkedList(LinkedList&& other) noexcept;

/\*\*

\* @brief - Оператор перемещения

\*/

LinkedList& operator=(const LinkedList& other);

/\*\*

\* @brief - Проверка на пустой список

\*/

bool IsEmpty() const;

/\*\*

\* @brief - Фунция преобразования в строку

\*/

std::string toString() const;

/\*\*

\* @brief - Функция вставки эллемента в конец списка

\*/

void push\_back(int value);

/\*\*

\* @brief - Функция вставки элемента в начало списка

\*/

void push\_front(int value);

/\*\*

\* @brief - Функция удаления элемента из конца списка

\*/

void pop\_back();

/\*\*

\* @brief - Функция удаление элемента из начала списка

\*/

void pop\_front();

/\*\*

\* @brief - Функция вставки эллемента по индексу

\*/

void insert(size\_t idx, int elem);

/\*\*

\* @brief - Функция удаления элемента по индексу

\*/

void remove(size\_t idx);

/\*\*

\* @brief - Оператор сдвига влево

\*/

friend std::ostream& operator«(std::ostream& os, const LinkedList& list);

};

}

//Файл List.cpp

#include "List.h"

#include <sstream>

rut\_miit::LinkedList::LinkedList() : head(nullptr), tail(nullptr) {}

rut\_miit::LinkedList::LinkedList(std::initializer\_list<int> initList) : LinkedList()

{

for (auto& value : initList)

{

push\_back(value);

}

}

rut\_miit::LinkedList::LinkedList(const LinkedList& other) : LinkedList()

{

Node\* temp = other.head;

while (temp != nullptr)

{

push\_back(temp->data);

temp = temp->next;

}

}

rut\_miit::LinkedList& rut\_miit::LinkedList::operator=(const LinkedList& other)

{

LinkedList temp(other);

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

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

return \*this;

}

rut\_miit::LinkedList::~LinkedList()

{

while (head != nullptr)

{

Node\* temp = head;

head = head->next;

delete temp;

}

}

bool rut\_miit::LinkedList::IsEmpty() const

{

return head == nullptr;

}

std::string rut\_miit::LinkedList::toString() const

{

std::stringstream result;

Node\* temp = head;

while (temp != nullptr)

{

result « temp->data « " ";

temp = temp->next;

}

return result.str();

}

void rut\_miit::LinkedList::push\_back(int value)

{

Node\* newNode = new Node{ value, nullptr, tail};

if (head == nullptr)

{

head = newNode;

}

else

{

tail->next = newNode;

}

tail = newNode;

}

void rut\_miit::LinkedList::push\_front(int value)

{

Node\* newNode = new Node{ value, head, nullptr };

if (head == nullptr)

{

tail = newNode;

}

else

{

head->prev = newNode;

}

head = newNode;

}

void rut\_miit::LinkedList::pop\_back()

{

if (tail == nullptr)

{

return;

}

if (head->next == nullptr)

{

delete tail;

head = nullptr;

tail = nullptr;

return;

}

Node\* temp = tail;

tail = tail->prev;

delete temp;

tail->next = nullptr;

}

void rut\_miit::LinkedList::pop\_front()

{

if (head == nullptr)

{

return;

}

if (head->next == nullptr)

{

delete head;

head = nullptr;

tail = nullptr;

return;

}

Node\* temp = head;

head = head->next;

delete temp;

head->prev = nullptr;

}

std::ostream& rut\_miit::operator«(std::ostream& os, const LinkedList& list)

{

os « list.toString();

return os;

}

void rut\_miit::LinkedList::insert(size\_t idx, int elem)

{

if (idx < 0) {

throw std::invalid\_argument("Wrong value index!");

}

size\_t index = idx;

Node\* current = head;

size\_t curr\_index = 0;

while (curr\_index < index)

{

current = current->next;

curr\_index++;

}

if (current == nullptr) {

push\_back(elem);

}

else {

Node\* tmp\_current = current;

Node\* tmp\_prev = current->prev;

current = new Node();

current->data = elem;

current->next = tmp\_current;

current->prev = tmp\_prev;

if (tmp\_current != nullptr) {

tmp\_current->prev = current;

}

if (tmp\_prev != nullptr) {

tmp\_prev->next = current;

}

else {

head = current;

}

}

}

void rut\_miit::LinkedList::remove(size\_t idx)

{

if (idx < 0) {

throw std::invalid\_argument("Wrong value index!");

}

if (idx == 0) {

pop\_front();

}

else {

size\_t index = idx;

Node\* current = head;

size\_t curr\_index = 0;

while (curr\_index < index)

{

current = current->next;

curr\_index++;

}

Node\* tmp\_next = current->next;

Node\* tmp\_prev = current->prev;

delete current;

if (tmp\_next != nullptr) {

tmp\_next->prev = tmp\_prev;

}

if (tmp\_prev != nullptr) {

tmp\_prev->next = tmp\_next;

}

}

}

rut\_miit::LinkedList::LinkedList(LinkedList&& other) noexcept

{

head = other.head;

tail = other.tail;

other.head = nullptr;

other.tail = nullptr;

}

//Файл main.cpp

#include <iostream>

#include "..//Solver/List.h"

/\*\*

\* @brief Точка входа в программу

\* @return 0 в случае успеха

\*/

int main()

{

std::setlocale(NULL, "ru\_RU.UTF-8");

rut\_miit::LinkedList list = { 1, 2, 3, 4 };

std::cout « "Первоначальный список: " « list « std::endl;

list.push\_front(5);  
std::cout « "Список после push\_front(5): " « list « std::endl;

list.push\_back(5);

std::cout « "Список после push\_back(5): " « list « std::endl;

list.pop\_front();

std::cout « "Список после pop\_front(): " « list « std::endl;

list.pop\_back();

std::cout « "Список после pop\_back(): " « list « std::endl;

list.insert(3, 10);

std::cout « "Список после insert " « list « std::endl;

list.remove(0);

std::cout « "Список после remove(3): " « list « std::endl;

return 0;

}

1.4 Результаты выполнения программы

Результаты выполнения программы в C++ представлена ниже (Рисунок 2).

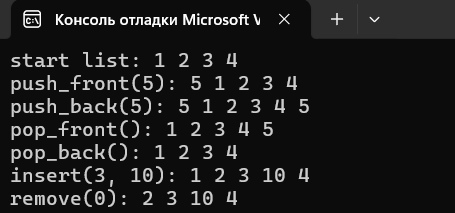


Рисунок 2 - Результаты выполнения программы

1.5 Выполнение тестовых примеров задание

Выполнены тестовые примеры. Результаты их выполнения представлены ниже

(Рисунок 3–5).

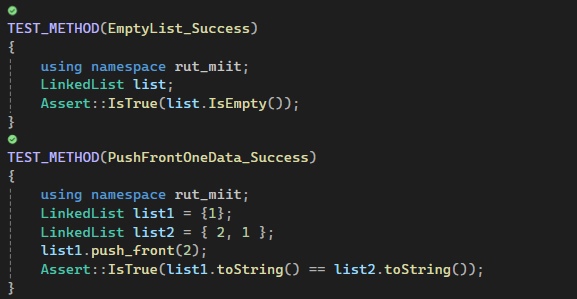


Рисунок 3 - Результат выполнения тестов

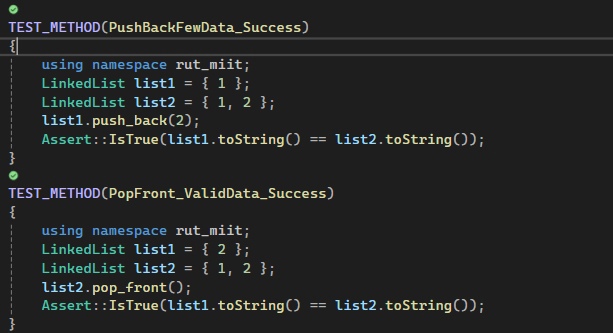


Рисунок 4 - Результат выполнения тестов

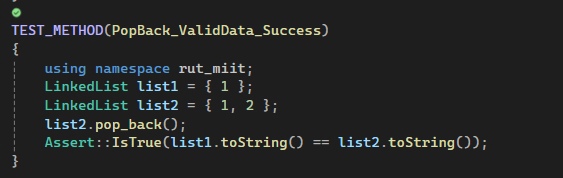


Рисунок 5 - Результат выполнения тестов

1.6 Отметка о выполнении задания в веб-хостинге системы контроля версий задание

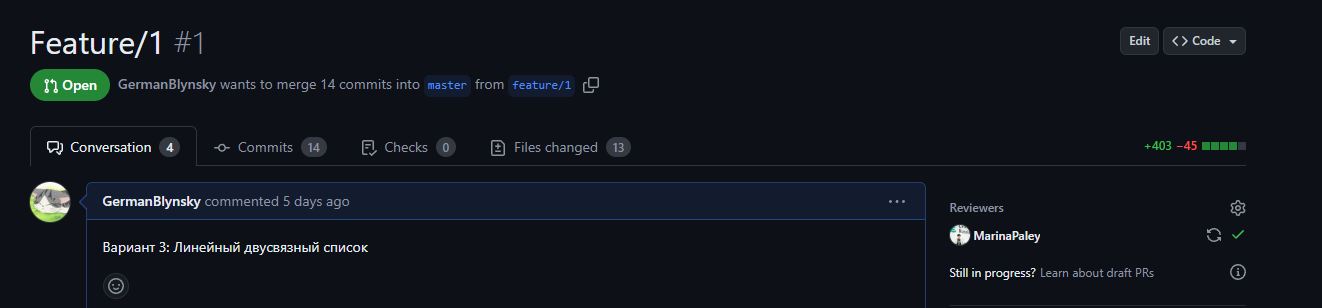


Рисунок 6 – Approve задачи