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

Хугаева М. М.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

Код:

#include <iostream>

#include "../Project1.1/Dequeue.hpp"

int main()

{

Dequeue<int> dequeue(2);

return 0;

}

#pragma once

#include <cmath>

#include <ostream>

#include <sstream>

#include <string>

template <typename Type>

class Dequeue {

public:

/\*\*

\*@brief Инициализирует класс Dequeue

\*/

Dequeue();

/\*\*

\*@brief Создает пустую ограниченную очередь, содержащую до `capacity`

\*элементов для `int` типов данных

\*

\*@param capacity Максимальный размер очереди

\*/

Dequeue(const int capacity);

/\*\*

\*@brief Удаляет объект

\*/

~Dequeue();

/\*\*

\*@brief Конструктор копирования

\*/

Dequeue(const Dequeue& deque);

/\*\*

\*@brief Конструктор перемещения

\*/

Dequeue(Dequeue&& deque) noexcept;

size\_t getSize();

/\*\*

\*@brief Удаляет элемент из передней части очереди

\*

\*@return Удалённый элемент из передней части очереди

\*/

Type takeFront();

/\*\*

\*@brief Удаляет элемент из задней части очереди

\*

\*@return Удалённый элемент из задней части очереди

\*/

Type takeBack();

/\*\*

\*@brief Получить элемент из передней части очереди

\*

\*@return Элемент из передней части очереди

\*/

Type getFront() const;

/\*\*

\*@brief Получить элемент из задней части очереди

\*

\*@return Элемент из задней части очереди

\*/

Type getBack() const;

/\*\*

\*@brief Записывает заданный элемент в начало очереди

\*

\*@param element Элемент, который должен быть помещен в начало очереди

\*/

void offerFront(const Type& element);

/\*\*

\*@brief Записывает заданный элемент в конец очереди

\*

\*@param element Элемент, который должен быть помещен в конец очереди

\*/

void offerBack(const Type& element);

/\*\*

\*@brief Получить является ли очередь пустой

\*

\*@return true Очередь пустая

\*@return false Очередь содержит некоторые элементы

\*/

bool isEmpty() const;

/\*\*

\* @brief Получить является ли очередь полной

\*

\*@return true Очередь полная

\*@return false Очередь не полна

\*/

bool isFull() const;

/\*\*

\*@brief Функция, которая преобразует коллекцию в строку.

\*@return Строку, состоящую из Nodes.

\*/

std::string ToString() noexcept;

/\*\*

\*@brief Функция, перегружающая оператор сравнения.

\*@param Сравнивает второй список.

\*@return Если содержимое списка равно второму списку то значение true и в противном случае значение false.

\*/

bool operator==(Dequeue<Type>& second);

/\*\*

\*@brief Замена переменных

\*/

friend void swap(Dequeue& lhs, Dequeue& rhs);

Dequeue& operator = (const Dequeue& Dequeue);

Dequeue& operator = (Dequeue&& Dequeue) noexcept;

void assertNotFull() const;

void assertNotEmpty() const;

private:

Type\* elements;

int start\_index;

int end\_index;

int capacity;

};

template <typename Type>

Dequeue<Type>::Dequeue()

:elements(nullptr), start\_index(0), end\_index(0), capacity(0)

{

}

template <typename Type>

inline Dequeue<Type>::Dequeue(const int capacity)

: start\_index(-1),

end\_index(0),

capacity(capacity),

elements(new Type[capacity])

{

if ( capacity <= 0)

{

throw std::logic\_error("Невожможное значение!");

}

}

template<typename Type>

inline Dequeue<Type>::~Dequeue()

{

delete[]elements;

}

template<typename Type>

inline Dequeue<Type>::Dequeue(const Dequeue& deque)

: start\_index(-1),

end\_index(0),

capacity(capacity),

elements(new int[capacity])

{

this->capacity = deque.capacity;

this->start\_index = deque.start\_index;

this->end\_index = deque.end\_index;

for (auto i = 0; i < capacity; ++i)

{

this->elements[i] = deque.elements[i];

}

}

template<typename Type>

inline Dequeue<Type>::Dequeue(Dequeue&& deque) noexcept : Dequeue()

{

std::swap(\*this, deque);

}

template<typename Type>

inline size\_t Dequeue<Type>::getSize()

{

return capacity;

}

;

template <typename Type>

inline Type Dequeue<Type>::takeFront() {

assertNotEmpty();

Type element = getFront();

if (start\_index == end\_index) {

start\_index = -1;

end\_index = -1;

}

else if (start\_index == capacity - 1)

start\_index = 0;

else

start\_index++;

return element;

};

template <typename Type>

inline Type Dequeue<Type>::takeBack() {

assertNotEmpty();

Type element = getBack();

if (start\_index == end\_index) {

start\_index = -1;

end\_index = -1;

}

else if (end\_index == 0)

end\_index = capacity - 1;

else

end\_index--;

return element;

};

template <typename Type>

inline Type Dequeue<Type>::getFront() const {

assertNotEmpty();

return elements[start\_index];

};

template <typename Type>

inline Type Dequeue<Type>::getBack() const {

assertNotEmpty();

return elements[end\_index];

};

template <typename Type>

inline void Dequeue<Type>::offerFront(const Type& element) {

assertNotFull();

if (start\_index == -1) {

start\_index = 0;

end\_index = 0;

}

else if (start\_index == 0)

start\_index = capacity - 1;

else

--start\_index;

elements[start\_index] = element;

};

template <typename Type>

inline void Dequeue<Type>::offerBack(const Type& element) {

assertNotFull();

if (start\_index == -1) {

start\_index = 0;

end\_index = 0;

}

else if (end\_index == capacity - 1)

end\_index = 0;

else

end\_index++;

elements[end\_index] = element;

};

template <typename Type>

inline bool Dequeue<Type>::isEmpty() const {

return start\_index == -1;

};

template <typename Type>

inline bool Dequeue<Type>::isFull() const {

return (start\_index == 0 && end\_index == capacity - 1) ||

start\_index == end\_index + 1;

}

template<typename Type>

inline std::string Dequeue<Type>::ToString() noexcept

{

std::stringstream buffer;

for (size\_t i = 0; i < capacity; i++)

{

buffer << elements[i] << ' ';

}

return buffer.str();

}

template<typename Type>

inline bool Dequeue<Type>::operator==(Dequeue<Type>& second)

{

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

}

template<typename Type>

inline Dequeue<Type>& Dequeue<Type>::operator=(const Dequeue& dequeue)

{

if (this == &dequeue)

{

return \*this;

}

std::swap(\*this, dequeue);

return \*this;

}

template<typename Type>

inline Dequeue<Type>& Dequeue<Type>::operator=(Dequeue&& dequeue) noexcept

{

if (\*this != dequeue)

{

std::swap(\*this, dequeue);

}

return \*this;

}

template<typename Type>

std::wstring ToString(const Dequeue<Type>& dequeue)

{

std::wstringstream buffer{};

buffer << dequeue;

return buffer.str();

}

template <typename Type>

inline void Dequeue<Type>::assertNotFull() const {

if (isFull()) throw std::length\_error("Очередь полная");

};

template <typename Type>

inline void Dequeue<Type>::assertNotEmpty() const {

if (isEmpty())

throw std::out\_of\_range("Очередь не содержит элементов");

};

Тесты:

#include "pch.h"

#include "CppUnitTest.h"

#include "../Project1.1/Dequeue.hpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace DequeueTests

{

TEST\_CLASS(DequeueTests)

{

public:

TEST\_METHOD(DequeueInt\_ValidData\_Success)

{

const size\_t size = 5;

auto dequeue = new Dequeue<int>(size);

Assert::IsNotNull(dequeue);

}

TEST\_METHOD(DequeueString\_ValidData\_Success)

{

const size\_t size = 5;

auto dequeue = new Dequeue<std::string>(size);

Assert::IsNotNull(dequeue);

}

TEST\_METHOD(TakeFront\_DequeueString\_ValidData)

{

const size\_t size = 5;

auto dequeue = new Dequeue<std::string>(size);

dequeue->offerFront("0");

Assert::IsTrue(dequeue->getFront() == "0");

}

TEST\_METHOD(TakeFront\_DequeueInt\_ValidData)

{

const size\_t size = 5;

auto dequeue = new Dequeue<int>(size);

dequeue->offerFront(0);

Assert::IsTrue(dequeue->getFront() == 0);

}

TEST\_METHOD(TakeBack\_DequeueString\_ValidData)

{

const size\_t size = 5;

auto dequeue = new Dequeue<std::string>(size);

dequeue->offerBack("0");

Assert::IsTrue(dequeue->getBack() == "0");

}

TEST\_METHOD(TakeBack\_DequeueInt\_ValidData)

{

const size\_t size = 5;

auto dequeue = new Dequeue<int>(size);

dequeue->offerBack(0);

Assert::IsTrue(dequeue->getBack() == 0);

}

TEST\_METHOD(ComparionOperatorInt\_ValidData\_Success)

{

Dequeue<int> dequeueone(10);

dequeueone.offerFront(1);

dequeueone.offerFront(2);

Dequeue<int> dequeuetwo(10);

dequeueone.offerFront(1);

dequeueone.offerFront(2);

Assert::IsFalse(dequeueone == dequeuetwo);

}

TEST\_METHOD(IsCollectionClearInt\_ValidData\_Success)

{

auto dequeue = new Dequeue<int>(1);

Assert::IsTrue(dequeue->isEmpty());

}

TEST\_METHOD(IsCollectionClearString\_ValidData\_Success)

{

auto dequeue = new Dequeue<std::string>(1);

Assert::IsTrue(dequeue->isEmpty());

}

TEST\_METHOD(IsCollectionFull\_Int\_ValidData\_Success) {

auto dequeue = new Dequeue<int>(1);

dequeue->offerFront(1);

Assert::IsTrue(dequeue->isFull());

}

TEST\_METHOD(IsCollectionFull\_String\_ValidData\_Success)

{

auto dequeue = new Dequeue<std::string>(1);

dequeue->offerFront("1");

Assert::IsTrue(dequeue->isFull());

}

TEST\_METHOD(MoveCtorIntDequeue\_ValidData\_Success)

{

Dequeue<int> d(10);

size\_t size = 10;

Dequeue<int> D = d;

Assert::IsTrue(d == D);

}

};

};

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

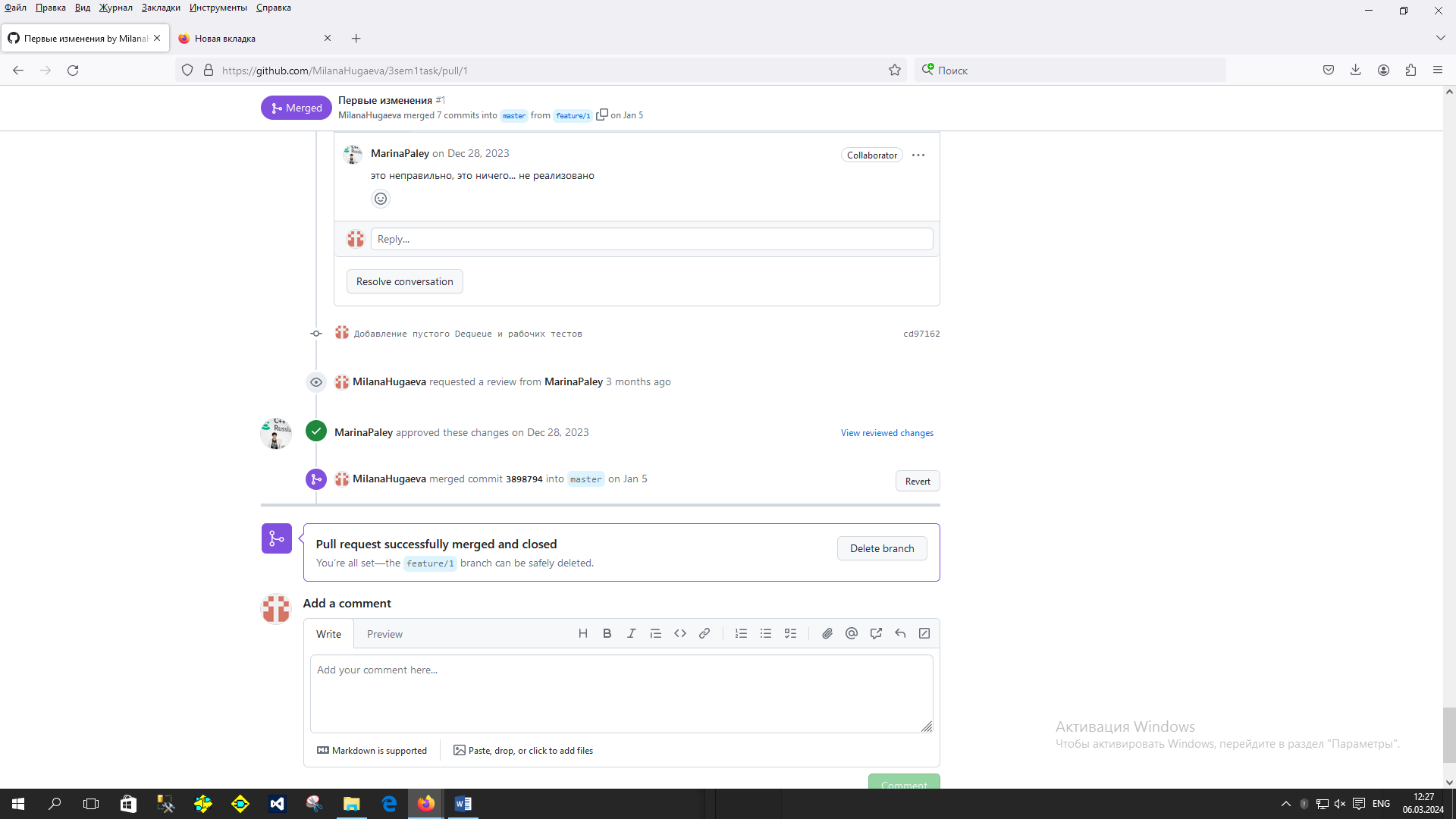
Автоматически созданное описание

**Рисунок 1 - Сводка работы тестов**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Рисунок 2 - UML-диаграмма**



**Рисунок 3 - Approve**