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

Москалев П.А

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

Задание 1

## Формулировка задачи

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. 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?).

**Вариант 2. Линейный односвязный список**

## UML - диаграмма алгоритма

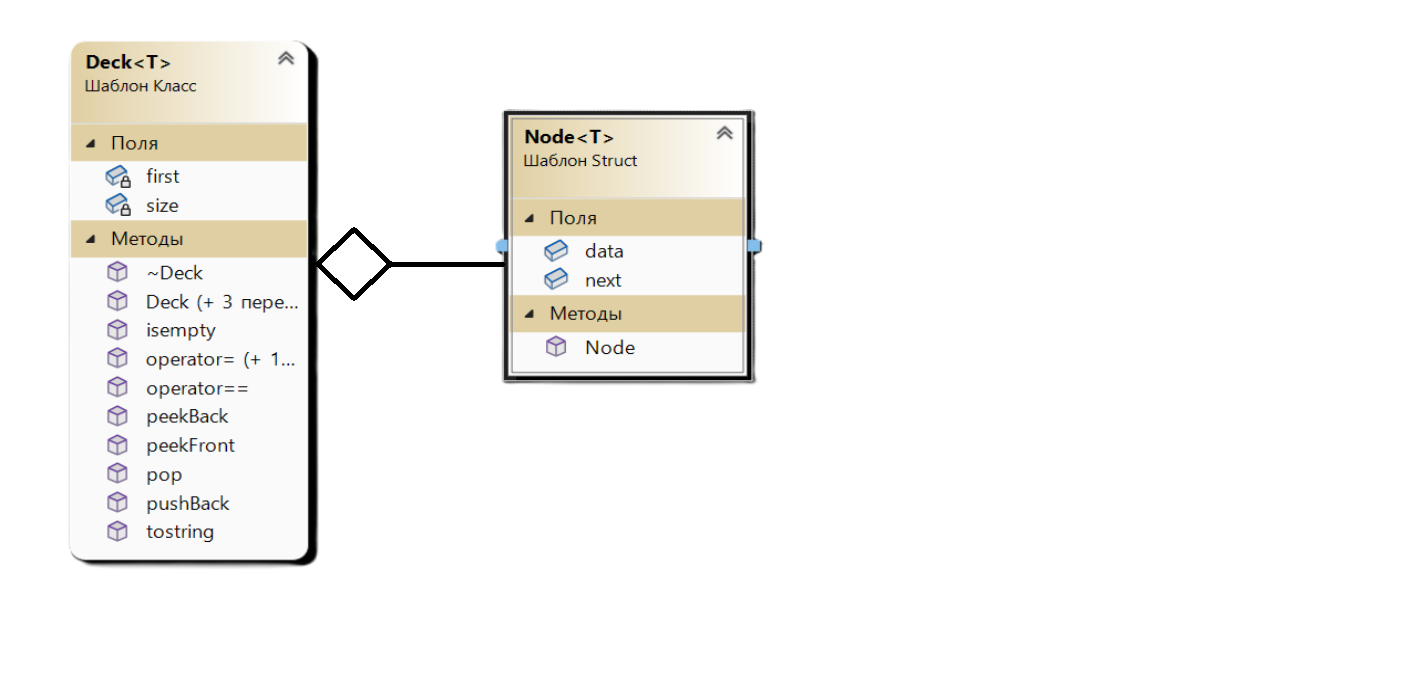


Рисунок 2 – UML диаграмма класса.

## Решение задачи на языке программирования C++

main.cpp:

#include "list.h"

int main()

{

Deck <int> list1;

list1.pushBack(2);

list1.pushBack(3);

std::cout<<list1.tostring();

Deck <std::string> list2;

list2.pushBack("qw");

list2.pushBack("wr");

Deck<int> list3(list1);

std::cout<<list2.tostring();

std::cout<<list2.peekBack();

}

List.h:

#pragma once

#include "node.h"

#include <iostream>

#include <string>

#include <sstream>

template <typename T>

class Deck{

public:

/\*\*

\* @brief Конструктор по умолчанию для деки.

\*

\* Конструктор инициализирует новую деку, устанавливая указатель на начальный

\* узел (first) в nullptr и размер деки (size) в ноль. Этот конструктор создает

\* пустую деку, готовую к добавлению элементов.

\*\*/

Deck();

/\*\*

\* @brief Конструктор инициализации деки с использованием списка инициализации.

\*

\* Конструктор создает новую деку, инициализируя ее содержимым переданного списка

\* инициализации `list`. Для каждого элемента в списке вызывается функция `pushBack`,

\* которая добавляет элемент в конец деки.

\*

\* @param list Список инициализации, содержащий элементы для инициализации деки.

\*\*/

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

/\*\*

\* @brief Деструктор деки.

\*

\* Деструктор освобождает динамически выделенную память, используемую для хранения

\* элементов деки. Путем вызова функции `pop()` в цикле, деструктор освобождает

\* ресурсы для каждого элемента до тех пор, пока размер деки не станет равным нулю.

\*

\*\*/

~Deck();

/\*\*

\* @brief Конструктор копирования для деки.

\*

\* Конструктор создает новую деку, глубоко копируя содержимое другой деки (`list2`).

\* Если дека `list2` не пуста, каждый элемент её копируется в новую деку.

\*

\* @param list2 Дека, содержимое которой копируется в новую деку.

\*

\*\*/

Deck(const Deck& list2);

/\*\*

\* @brief Перегрузка оператора перемещения для деки.

\*

\* Функция выполняет перемещение содержимого другой деки (`list2`) в текущую деку.

\* Операция перемещения более эффективна, чем глубокое копирование, поскольку

\* осуществляется обмен указателями, без фактического копирования данных.

\*

\* @param list2 R-value ссылка на деку, содержимое которой перемещается.

\*

\* @return Ссылка на текущую деку после перемещения.

\*\*/

Deck(Deck&& list2) noexcept;

/\*\*

\* @brief Добавление элемента в конец деки.

\*

\* Функция добавляет новый элемент с указанным значением в конец деки. Если дека

\* пуста, создается новый узел и устанавливается как начальный узел деки. Если дека

\* не пуста, функция перемещается к концу деки и добавляет новый узел.

\*

\* @param data Значение элемента, который добавляется в конец деки.

\*\*/

void pushBack (const T & data);

/\*\*

\* @brief Проверка на пустоту деки.

\*

\* Функция возвращает логическое значение, указывающее, является ли дека пустой.

\*

\* @return true, если дека пуста, и false в противном случае.

\*\*/

bool isempty ();

/\*\*

\* @brief Получение строкового представления деки.

\*

\* Функция создает строку, представляющую содержимое деки в виде пробел-separated

\* последовательности элементов. Если дека пуста, возвращается пустая строка.

\*

\* @return Строковое представление содержимого деки.

\*\*/

std::string tostring();

/\*\*

\* @brief Перегрузка оператора сравнения для дек.

\*

\* Функция сравнивает текущую деку с другой декой (`list2`) на основе строкового представления

\* каждой деки. Деки считаются равными, если их строковые представления совпадают.

\*

\* @param list2 Дека, с которой выполняется сравнение.

\*

\* @return true, если деки равны, и false в противном случае.

\*\*/

bool operator==(Deck& list2);

/\*\*

\* @brief Перегрузка оператора копирования для деки.

\*

\* Функция выполняет глубокое копирование содержимого другой деки (`list2`) в текущую деку.

\* Если дека содержит динамически выделенные ресурсы, они будут корректно скопированы,

\* чтобы обеспечить независимость между деками после присваивания.

\*

\* @param list2 Константная ссылка на деку, содержимое которой копируется в текущую деку.

\*

\* @return Ссылка на текущую деку после копирования.

\*\*/

Deck& operator=(const Deck& list2);

/\*\*

\* @brief Перегрузка оператора перемещения для деки.

\*

\* Функция выполняет перемещение содержимого другой деки (`list2`) в текущую деку.

\* Операция перемещения более эффективна, чем глубокое копирование, поскольку

\* осуществляется обмен указателями, без фактического копирования данных.

\*

\* @param list2 R-value ссылка на деку, содержимое которой перемещается.

\*

\* @return Ссылка на текущую деку после перемещения.

\*\*/

Deck& operator=(Deck&& list2) noexcept;

/\*\*

\* @brief Извлекает и удаляет элемент из начала деки.

\*

\* Функция предназначена для извлечения и удаления элемента из начала деки.

\* После успешного выполнения функции, размер деки уменьшается на единицу,

\* и освобождается память, занимаемая удаленным элементом.

\*\*/

void pop();

/\*\*

\* @brief Возвращает значение элемента, находящегося в начале деки, без его удаления.

\*

\* Функция предназначена для получения значения элемента, находящегося в начале деки,

\* без изменения самой деки. Это полезно, когда необходимо проверить значение

\* в начале деки без извлечения или изменения её содержимого.

\*

\* @return Значение элемента в начале деки.

\*\*/

T peekFront ();

/\*\*

\* @brief Возвращает значение элемента, находящегося в конце деки, без его удаления.

\*

\* Функция предназначена для получения значения элемента, находящегося в конце деки,

\* без изменения самой деки. Это полезно, когда необходимо проверить значение

\* в конце деки без извлечения или изменения её содержимого.

\*

\* @return Значение элемента в конце деки.

\*\*/

T peekBack ();

private:

Node <T>\* first;

size\_t size;

};

template <typename T>

T Deck<T>::peekFront ()

{

return first->data;

}

template <typename T>

T Deck<T>::peekBack ()

{

Node <T>\* last=first;

for (Node <T>\* node=this->first; node != nullptr; node=node->next)

{

last = node;

}

return last->data;

}

template <typename T>

Deck<T>::Deck():

first(nullptr), size(0)

{

}

template <typename T>

Deck<T>::Deck(std::initializer\_list<T> list):

first (nullptr), size (0)

{

for(auto& item : list)

{

pushBack(item);

}

}

template <typename T>

Deck<T>::~Deck()

{

while (size != 0)

{

pop ();

}

}

template <typename T>

Deck<T>::Deck(const Deck& list2)

{

if (this->first==nullptr)

{

for (Node <T>\* node=list2.first; node != nullptr; node=node->next)

{

this->pushBack(node->data);

}

}

}

template <typename T>

void Deck<T>::pushBack (const T & data)

{

if (first==nullptr)

{

first=new Node <T> (data);

}

else

{

Node <T>\* thisNode=first;

while (thisNode->next!=nullptr)

{

thisNode=thisNode->next;

}

thisNode->next=new Node <T> (data);

}

size++;

}

template <typename T>

bool Deck<T>::isempty()

{

return size==0;

}

template <typename T>

std::string Deck<T>::tostring()

{

std::stringstream buffer;

if (first != nullptr)

{

Node<T>\* thisNode = first;

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

{

buffer << thisNode->data << " ";

thisNode = thisNode->next;

}

}

return buffer.str();

}

template <typename T>

bool Deck<T>::operator==(Deck& list2)

{

return (this->tostring()==list2.tostring());

}

template <typename T>

Deck<T> & Deck<T>::operator=(const Deck& list2)

{

if (this!=list2)

{

Deck<T> tempo (list2);

std::swap(tempo.first, this->first);

std::swap(tempo.size, this->size);

}

return\*this;

}

template <typename T>

void Deck<T>::pop()

{

Node <T>\*tempo=first;

first=first->next;

size--;

delete tempo;

}

template <typename T>

Deck<T>& Deck<T>::operator=(Deck&& list2) noexcept

{

std::swap(this->first, list2.first);

return \*this;

}

Node.h:

#pragma once

template <typename T>

struct Node {

Node (T data, Node\* next=nullptr);

T data;

Node\* next;

};

template <typename T>

Node <T>::Node(T data, Node\* next):

data(data), next(next)

{

}

## Решение тестов

Код тестов:

#include "pch.h"

#include "CppUnitTest.h"

#include"..//Project1/list.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace DeckTest

{

TEST\_CLASS(DeckTest)

{

public:

TEST\_METHOD(DeckCopyConstructorSuccess)

{

Deck<int> d1;

d1.pushBack(2);

d1.pushBack(3);

Deck<int> d2(d1);

bool AreEqual = d1 == d2;

Assert::IsTrue(AreEqual);

}

TEST\_METHOD(DeckCopyOperatorSuccess)

{

Deck<int> d1;

d1.pushBack(2);

d1.pushBack(3);

Deck<int> d2 = d1;

bool AreEqual = d1 == d2;

Assert::IsTrue(AreEqual);

}

TEST\_METHOD(DeckPushBackSuccess)

{

Deck<int> d1;

d1.pushBack(2);

d1.pushBack(3);

std::string realstring = d1.tostring();

std::string Expstring = "2 3 ";

bool AreEqual = realstring == Expstring;

Assert::IsTrue(AreEqual);

}

TEST\_METHOD(DeckIsEmptySuccess)

{

Deck<int> d1;

d1.pushBack(2);

d1.pushBack(3);

bool isEmpty = d1.isempty();

Assert::IsFalse(isEmpty);

}

TEST\_METHOD(DeckComparisonOperatorSuccess)

{

Deck<int> d1;

Deck<int> d2;

d1.pushBack(2);

d1.pushBack(3);

d2.pushBack(2);

d2.pushBack(3);

bool AreEqual = d1== d2;

Assert::IsTrue(AreEqual);

}

TEST\_METHOD(DeckPopSuccess)

{

Deck<int> d1;

Deck<int> d2;

d1.pushBack(2);

d1.pushBack(3);

d2.pushBack(2);

d2.pushBack(3);

d2.pop();

bool AreEqual = d1 == d2;

Assert::IsFalse(AreEqual);

}

TEST\_METHOD(DeckPeekFrontSuccess)

{

Deck<int> d1;

d1.pushBack(3);

d1.pushBack(5);

int fr = d1.peekFront();

bool AreEqual = fr == 3;

Assert::IsTrue(AreEqual);

}

TEST\_METHOD(DeckPeekBackSuccess)

{

Deck<int> d1;

d1.pushBack(3);

d1.pushBack(5);

int fr = d1.peekBack();

bool AreEqual = fr == 5;

Assert::IsTrue(AreEqual);

}

};

}

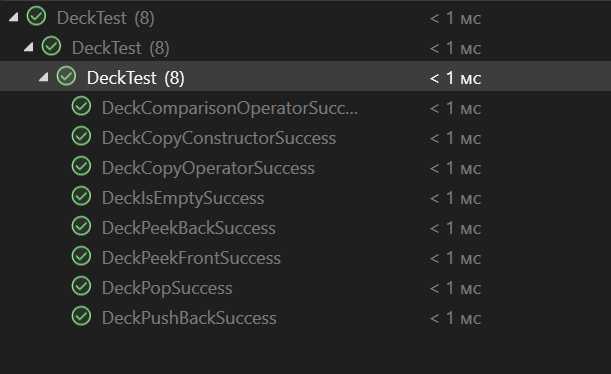


Рисунок 4 – Успешные тесты.