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 № 2

По дисциплине «Языки программирования»

Выполнил: ст. гр. ТКИ – 241

Белавинская М.А.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

Написать класс вектор с использованием темплейтов

**Код задания:**

﻿#pragma once

#include <initializer\_list>

#include <ostream>

#include <iostream>

#include <sstream>

template <typename T>

class Vector final

{

public:

/\*

@brief Инициализация нового экземпляра типа vector с помощью initializer\_list

@param list Лист с помощью которго будет создан vector

\*/

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

/\*

@brief Удаляет, чистит память после вектора

\*/

~Vector();

/\*

@brief Получение колличества элементов в векторе

@return Колличество элементов в векторе

\*/

size\_t getSize();

/\*

@brief Удаление элемента из вектора по индексу элемента

@param index Индекс элемента

\*/

void deleteI(size\_t index);

/\*

@brief Получение индекса элемента в векторе

@param value Элемент который мы ищем

@return Индекс элемента в векторе, вернет -1, если элемента в векторе нет

\*/

int findI(const T& value);

/\*

@brief Проверка на отсутствие элементов в векторе

@return true если элементов нет и false если они есть

\*/

bool isEmpty() const;

/\*

@brief Перегрузка оператора квадратных скобок

@param index Индекс элемента в векторе

@return Элемент из вектора по заданному индексу

\*/

T& operator[](size\_t index) const;

/\*

@brief Перегрузка оператора копирования

@param Vector Вектор который будет скопирован

@return Скопированный Vector

\*/

Vector& operator=(const Vector<T>& vector);

/\*

@brief Перегрузка оператора перемещения

@param Vector Вектор для перемещения

@return Вектор

\*/

Vector& operator=(Vector<T>&& vector) noexcept;

/\*

@brief Перегрузка конструктора копирования

@param vector Вектор для перемещния

\*/

Vector(const Vector& vector);

/\*

@brief Перегрузка конструктор перемещ

@param vector Вектор для копирования

\*/

Vector(Vector<T>&& vector) noexcept;

/\*

@brief Перегрузка оператора вывода

@param os Поток вывода

@param vector Вектор для вывода

\*/

friend std::ostream& operator<<(std::ostream& os, Vector<T>& vector);

/\*

@brief Перегрузка оператора равно

@param rha Первый аргумент для сравнения

@param lha Второй аргумент для сравнения

@return false - если равны, true - если не равны

\*/

bool operator==(Vector& second);

/\*

@brief Перегрузка оператора не равно

@param rha Первый аргумент для сравнения

@param lha Второй аргумент для сравнения

@return true - если равны, false - если не равны

\*/

bool operator!=(Vector& second);

/\*

@brief Преобразование объета в строку

@return Строка из объекта

\*/

std::string toString() const noexcept;

private:

size\_t size;

T\* array;

};

template <typename T>

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

:size(list.size())

{

this->array = new int[this->size];

std::copy(list.begin(), list.end(), this->array);

}

template <typename T>

inline Vector<T>::~Vector()

{

delete[] this->array;

}

template<typename T>

inline size\_t Vector<T>::getSize()

{

return this->size;

}

template <typename T>

inline bool Vector<T>::isEmpty() const

{

return this->size == 0;

}

template <typename T>

inline Vector<T>& Vector<T>::operator=(const Vector& vector)

{

if (vector == \*this)

{

return \*this;

}

if (!this->isEmpty())

{

delete[] this->array;

this->array = nullptr;

this->size = 0;

}

Vector<T> temp(vector);

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

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

return \*this;

}

template <typename T>

inline Vector<T>&Vector<T>::operator=(Vector&& vector) noexcept

{

std::swap(vector.array, this->array);

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

return \*this;

}

template <typename T>

inline Vector<T>::Vector(const Vector& vector)

:size(vector.getSize()), array(nullptr)

{

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

{

array.push

}

}

template <typename T>

inline Vector<T>::Vector(Vector&& vector) noexcept

:size(0), array(nullptr)

{

std::swap(vector.array, this->array);

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

}

template <typename T>

inline T&Vector<T>::operator[](size\_t index) const

{

if (this->isEmpty() || index >= this->size or index < 0)

{

throw std::out\_of\_range("Incorrect Index");

}

return this->array[index];

}

template <typename T>

inline void Vector<T>::deleteI(size\_t index)

{

if (this->size == 0 || index >= this->size)

{

throw std::out\_of\_range("Incorrect Index");

}

T\* vector1 = new T[this->size - 1];

T now = 0;

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

{

if (i != index)

{

vector1[now] = this->array[i];

now++;

}

}

delete[] this->array;

this->array = vector1;

this->size--;

}

template <typename T>

inline int Vector<T>::findI(const T& value)

{

if (this->isEmpty())

{

return -1;

}

size\_t index = 0;

while ((index < this->size) && this->array[index] != value)

{

index++;

}

if (index == this->size)

{

return -1;

}

else

{

return index;

}

}

template<typename T>

inline bool Vector<T>::operator==(Vector& second)

{

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

}

template <typename T>

inline std::string Vector<T>::toString() const noexcept

{

std::stringstream sstring;

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

{

sstring << this->array[i] << " ";

}

return sstring.str();

}

template <typename T>

inline std::ostream& operator<<(std::ostream& os, Vector<T>& vector)

{

return os << vector.toString();

}

template <typename T>

inline bool Vector<T>::operator!=(Vector& second)

{

return !(this->toString() == second.toString());

}

#include "Vector.h"

**Тесты:**

#include "pch.h"

#include "CppUnitTest.h"

#include"../VectorOnTemplates/Vector.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace TestVector

{

TEST\_CLASS(TestVector)

{

public:

TEST\_METHOD(OperatorForCompareTest\_ValitData\_Success)

{

Vector<int> vector1 {1, 4, 2, 6};

Vector<int> vector2 {1, 4, 2, 6};

bool are\_equal = (vector1 == vector2);

Assert::IsTrue(are\_equal);

}

TEST\_METHOD(ToString\_ValitData\_Success)

{

Vector<int> vector{1, 2, 3, 4};

bool are\_equal = (vector.toString() == "1 2 3 4 ");

Assert::IsTrue(are\_equal);

}

TEST\_METHOD(OperatorForNotCompareTest\_ValitData\_Success)

{

Vector<int> vector1 {1, 4, 2, 6};

Vector<int> vector2 {2, 4, 5, 6};

bool arequal = (vector1 != vector2);

Assert::IsTrue(arequal);

}

TEST\_METHOD(GetSize\_ValitData\_Success)

{

Vector<int> vector1 {1, 4, 2, 6};

bool arequal = (vector1.getSize()==4);

Assert::IsTrue(arequal);

}

TEST\_METHOD(DeletI\_ValitData\_Success)

{

Vector<int> vector1 {1, 4, 2 };

vector1.deleteI(1);

bool arequal = (vector1.toString()=="1 2 ");

Assert::IsTrue(arequal);

}

};

}

**UML Диаграмма:**

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

Автоматически созданное описание**

Рисунок 1 – UML диаграмма

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

Автоматически созданное описание**

Рисунок 2 – Approve