|  |  |
| --- | --- |
| **=Российский университет транспорта (МИИТ)**  **Институт транспортной техники и систем управления**  **Кафедра «Управление и защита информации»** | |
| **Отчет**  **по практическому заданию №2**  **по дисциплине «Системы управления базами данных»** | |
|  | Выполнил:  Студент группы ТКИ-442  Пшеницын П.Е. |
|  | Проверил:  к.т.н., доцент  Васильева М.А. |
| Москва 2023 | |

**Оглавление**

[1 Задание 3](#_Toc153716739)

[2 UML-диаграмма 4](#_Toc153716740)

[3 Код программы 5](#_Toc153716741)

[3.1 Программа 5](#_Toc153716742)

[3.1.1 Stack.h 5](#_Toc153716743)

[3.1.2 Stack.cpp 6](#_Toc153716744)

[3.2 Тесты 9](#_Toc153716745)

[3.2.1 StackTests.cpp 9](#_Toc153716746)

[4 Результаты выполнения тестов 12](#_Toc153716747)

1. Задание

Разработать структуру данных на языке программирования С++ в ООП парадигме (**Стек**). Реализовать библиотеку классов, обеспечить выполнение операций CRUD (create, read, update, delete) для любого типа данных (template). Ни один метод не должен зависеть от консоли или файла, используйте представление в строку.

Написать итератор для работы с коллекцией (разработанной структурой данных).

Все методы покрыть тестами (модульное тестирование).

Задание выполнять в системе контроля версий. Проверка задания осуществляется посредством pull request в веб-хостинге GitHub (collaborator MarinaPaley). В репозитории необходимо иметь рабочий код, так, чтобы для проверки преподаватель смог склонировать репозиторий и запустить тесты, не прибегая к помощи специальных инструментов.

1. UML-диаграмма

|  |
| --- |
|  |
| 1. – UML-диаграмма |

1. Код программы
   1. Программа
      1. Stack.h

#pragma once

#include <string>

#include <sstream>

#include <vector>

using namespace std;

/\*\*

\* @brief Элемент стека

\* @tparam T тип данных

\*/

template <typename T>

struct Node {

/\*\*

\* @brief Значение

\*/

T data;

/\*\*

\* @brief Указатель на предыдущий элемент стека

\*/

Node<T>\* prev;

/\*\*

\* @brief Параметризированный конструктор

\* @param data данные объекта

\*/

Node(const T& data);

};

/\*\*

\* @brief Класс Stack.

\* @tparam T тип данных.

\*/

template <typename T>

class Stack {

private:

/\*\*

\* @brief Указатель на вершину стека

\*/

Node<T>\* head;

/\*\*

\* @brief Размер стека.

\*/

size\_t size;

/\*\*

\* @brief Получение значений стека (с вершины). Нужен для копирования

\*/

vector<T> create\_values() const noexcept;

public:

/\*\*

\* @brief Конструктор по умолчанию.

\*/

Stack();

/\*\*

\* @brief Конструктор копирования.

\* @param obj объект Stack.

\*/

Stack(const Stack<T>& obj);

/\*\*

\* @brief Конструктор перемещения.

\* @param moved перемещаемый объект Stack.

\*/

Stack(Stack<T>&& moved) noexcept;

/\*\*

\* @brief Оператор копирования.

\* @param obj объект Stack.

\*/

Stack<T>& operator=(const Stack<T>& obj);

/\*\*

\* @brief Оператор перемещения.

\* @param moved объект Stack.

\*/

Stack<T>& operator=(Stack<T>&& moved) noexcept;

/\*\*

\* @brief Деструктор.

\*/

~Stack();

/\*\*

\* @brief Добавление элемента в стек.

\* @param item добавляемый объект.

\*/

void push(const T& item);

/\*\*

\* @brief Вывод стека на экран.

\*/

string show() const noexcept;

/\*\*

\* @brief Проверка наличия элементов в стеке.

\*/

bool is\_empty() const noexcept;

/\*\*

\* @brief Удаление последнего добавленного элемента.

\*/

void pop();

/\*\*

\* @brief Получение вершины стека.

\*/

T top() const;

/\*\*

\* @brief Получение размера стека.

\*/

size\_t get\_size() const noexcept;

};

* + 1. Stack.cpp

#include <iostream>

#include <string>

#include <sstream>

#include "Stack.h"

using namespace std;

template<typename T>

inline Node<T>::Node(const T& data) : data{ data }, prev{ nullptr } {}

template<typename T>

vector<T> Stack<T>::create\_values() const noexcept {

vector<T> values;

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

while (p != nullptr) {

values.push\_back(p->data);

p = p->prev;

}

return values;

}

template<typename T>

Stack<T>::Stack() : head{ nullptr }, size{ 0 } {}

template<typename T>

Stack<T>::Stack(const Stack<T>& obj) : Stack() {

if (!obj.is\_empty()) {

vector<T> values = obj.create\_values();

for (int i = values.size() - 1; i >= 0; --i) {

this->push(values[i]);

}

}

}

template<typename T>

Stack<T>::Stack(Stack<T>&& moved) noexcept : Stack() {

\*this = move(moved);

}

template<typename T>

Stack<T>& Stack<T>::operator=(const Stack<T>& obj) {

if (this != &obj) {

Stack<T> t{ obj };

\*this = move(t);

}

return \*this;

}

template<typename T>

Stack<T>& Stack<T>::operator=(Stack<T>&& moved) noexcept {

if (this != &moved) {

swap(this->head, moved.head);

swap(this->size, moved.size);

}

return \*this;

}

template<typename T>

Stack<T>::~Stack() {

while (!this->is\_empty()) {

this->pop();

}

}

template<typename T>

void Stack<T>::push(const T& item) {

Node<T>\* node = new Node<T>{ item };

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

node->prev = prev;

this->head = node;

++this->size;

}

template<typename T>

string Stack<T>::show() const noexcept {

ostringstream buffer;

if (this->is\_empty()) {

buffer << "Пустой стек\n";

}

else {

buffer << "{ ";

vector<T> values = this->create\_values();

for (int i = 0; i < values.size(); ++i) {

buffer << values[i] << ' ';

}

buffer << "}";

}

return buffer.str();

}

template<typename T>

bool Stack<T>::is\_empty() const noexcept {

return size == 0;

}

template<typename T>

void Stack<T>::pop() {

if (this->is\_empty()) {

throw logic\_error("Стек уже пуст");

}

T item = this->head->data;

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

this->head = this->head->prev;

delete t;

--this->size;

}

template<typename T>

T Stack<T>::top() const {

if (this->is\_empty()) {

throw logic\_error("Стек пуст");

}

return this->head->data;

}

template<typename T>

size\_t Stack<T>::get\_size() const noexcept {

return this->size;

}

template Node<int>;

template Node<string>;

template Stack<int>;

template Stack<string>;

* 1. Тесты
     1. StackTests.cpp

﻿#include "pch.h"

#include "CppUnitTest.h"

#include "../Stack/Stack.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace StackTests {

TEST\_CLASS(NodeTests) {

public:

TEST\_METHOD(CtorParam\_NoException) {

Node<int> node{ 3 };

Assert::IsNotNull(&node);

}

};

TEST\_CLASS(StackTests) {

public:

TEST\_METHOD(Ctor\_NoException) {

Stack<int> s{};

Assert::IsNotNull(&s);

}

TEST\_METHOD(CtorCopy\_NoException) {

Stack<int> s{};

s.Push(3);

Stack<int> copied{ s };

Assert::IsNotNull(&copied);

Assert::AreEqual(copied.GetSize(), s.GetSize());

Assert::AreEqual(copied.Top(), s.Top());

}

TEST\_METHOD(CtorMove\_NoException) {

Stack<int> s{};

s.Push(3);

size\_t expectedSize = s.GetSize();

int expectedTop = s.Top();

Stack<int> copied{ move(s) };

Assert::IsNotNull(&copied);

Assert::AreEqual(copied.GetSize(), expectedSize);

Assert::AreEqual(copied.Top(), expectedTop);

}

TEST\_METHOD(OperatorCopy\_NoException) {

Stack<int> s{};

s.Push(3);

Stack<int> copied = s;

Assert::AreEqual(copied.GetSize(), s.GetSize());

Assert::AreEqual(copied.Top(), s.Top());

}

TEST\_METHOD(OperatorMove\_NoException) {

Stack<int> s{};

s.Push(3);

size\_t expectedSize = s.GetSize();

int expectedTop = s.Top();

Stack<int> copied = move(s);

Assert::AreEqual(copied.GetSize(), expectedSize);

Assert::AreEqual(copied.Top(), expectedTop);

}

TEST\_METHOD(Int\_Push\_NoException) {

Stack<int> s{};

size\_t size = 1;

int value = 3;

s.Push(value);

Assert::AreEqual(size, s.GetSize());

Assert::AreEqual(value, s.Top());

}

TEST\_METHOD(String\_Push\_NoException) {

Stack<string> s{};

size\_t size = 1;

string value = "3";

s.Push(value);

Assert::AreEqual(size, s.GetSize());

Assert::AreEqual(value, s.Top());

}

TEST\_METHOD(Show\_Empty) {

Stack<int> s{};

string expectedString = "Пустой стек\n";

Assert::AreEqual(s.Show(), expectedString);

}

TEST\_METHOD(Show\_NotEmpty) {

Stack<int> s{};

s.Push(3);

s.Push(3);

s.Push(9);

string expectedString = "{ 9 3 3 }";

Assert::AreEqual(s.Show(), expectedString);

}

TEST\_METHOD(IsEmpty\_Empty) {

Stack<int> s{};

Assert::IsTrue(s.IsEmpty());

}

TEST\_METHOD(IsEmpty\_NotEmpty) {

Stack<int> s{};

s.Push(3);

Assert::IsFalse(s.IsEmpty());

}

TEST\_METHOD(Pop\_Exception) {

Assert::ExpectException<std::logic\_error>([]() {

Stack<int> s{};

s.Pop();

});

}

TEST\_METHOD(Pop\_NoException) {

Stack<int> s{};

s.Push(3);

s.Pop();

Assert::IsTrue(s.IsEmpty());

}

TEST\_METHOD(Top\_Exception) {

Assert::ExpectException<std::logic\_error>([]() {

Stack<int> s{};

s.Top();

});

}

TEST\_METHOD(Top\_NoException) {

Stack<int> s{};

s.Push(3);

int expectedTop = 3;

Assert::AreEqual(s.Top(), expectedTop);

}

TEST\_METHOD(GetSize\_Empty) {

Stack<int> s{};

size\_t expectedSize = 0;

Assert::AreEqual(s.GetSize(), expectedSize);

}

TEST\_METHOD(GetSize\_NotEmpty) {

Stack<int> s{};

s.Push(3);

size\_t expectedSize = 1;

Assert::AreEqual(s.GetSize(), expectedSize);

}

};

}

1. Результаты выполнения тестов

|  |
| --- |
|  |
| 1. – Прохождение тестов |