Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Кафедра ПМиК

**Лабораторная работа № 11**

**по дисциплине «Современные технологии программирования»**

**«Параметризованный абстрактный тип данных “Память”»**

Выполнил: студент 4 курса ф. ИВТ, гр. ИП-813

Пещеров Вячеслав Александрович

Проверил: ассистент

к. ПМиК Агалаков А.А.

Новосибирск, 2021

Оглавление

[**Задание** 3](#_Toc90051377)

[**Реализация** 7](#_Toc90051378)

[**Демонстрация работы** 8](#_Toc90051379)

[**Вывод** 9](#_Toc90051380)

[**Список литературы** 10](#_Toc90051381)

[**Приложение** 11](#_Toc90051382)

[Листинг 1. TMemory.h 11](#_Toc90051383)

[Листинг 2. TMemory.cpp 11](#_Toc90051384)

[Листинг 3. TMemoryUnitTests.cpp 13](#_Toc90051385)

# **Задание**

1. В соответствии с приведенной ниже спецификацией реализовать параметризованный абстрактный тип данных «память», для хранения одного числа – объекта типа T, используя шаблон классов *С++*.
2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования.
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

Спецификация типа данных «память». ADT **TMemory**

**Данные**

Память (тип TMemory, в дальнейшем - память) - это память для хранения

«числа» объекта типа T в поле FNumber, и значения «состояние памяти» в поле FState. Объект память - изменяемый. Он имеет два состояния, обозначаемых значениями: «Включена» (\_On), «Выключена» (\_Off). Её изменяют операции: Записать (Store), Добавить (Add), Очистить (Clear).

Операции

|  |  |
| --- | --- |
| ***Конструктор*** |  |
| Начальные значения: | Нет. |
| Процесс: | Инициализирует поле FNumber объекта  «память» (тип TMemory) объектом «число» (тип T) со значением по умолчанию. |

|  |  |
| --- | --- |
|  | Например для числа типа TFrac со значением 0/1. Память устанавливается в состояние «Выключена», в поле FState  «состояние памяти» заносится значение  (\_Off). |
| ***Записать*** |  |
| Вход: | (E) – объект тип T. |
| Предусловия: | Нет. |
| Процесс: | В объект «память» (тип TMemory) в поле FNumber записывается копия объекта Е. Память устанавливается в состояние  «Включена», в поле FState «состояние  памяти» заносится значение (\_On). |
| Выход: | Нет. |
| Постусловия: | Состояние памяти поле FState –  «Включена» (\_On). |
|  |  |
| ***Взять*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Создаёт и возвращает копию объекта хранящегося в объекте «память» (тип  TMemory) в поле FNumber. |
| Выход: | Объект типа T. |
| Постусловия: | Состояние памяти поле FState –  «Включена» (\_On). |
|  |  |
| ***Добавить*** |  |
| Вход: | (Е) – число объект типа T. |

|  |  |
| --- | --- |
| Предусловия: | Нет. |
| Процесс: | В поле FNumber объекта «память» (тип TMemory) записывается объект типа T, полученный в результате сложения числа (Е) и числа, хранящегося в памяти в поле  FNumber. |
| Выход: | Нет. |
| Постусловия: | Состояние памяти поле FState –  «Включена» (\_On). |
|  |  |
| ***Очистить*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | В поле числа (FNumber) объекта «память» (тип TMemory) записывается объект типа T со значением по умолчанию. Например, для простой дроби - 0/1. Память (поле FState) устанавливается в состояние «Выключена»  (\_Off). |
| Выход: | Нет. |
| Постусловия: | Состояние памяти поле FState –  «Выключена» (\_Off). |
|  |  |
| ***ЧитатьСостояниеПамяти*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Копирует и возвращает значение поля FState «состояние памяти» объекта  «память» (тип TMemory) в формате строки. |

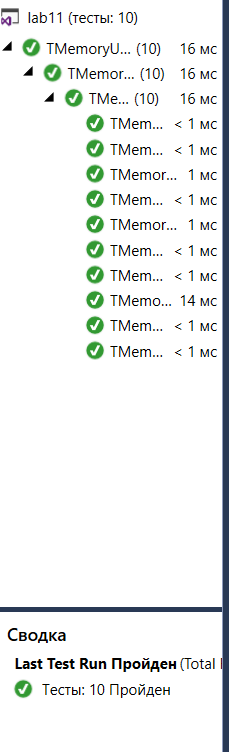
|  |  |
| --- | --- |
| Выход: | Значение поля «состояния памяти» (типа  String). |
| Постусловия: | Нет. |
|  |  |
| ***ЧитатьЧисло*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Копирует и возвращает значение поля  «число» (FNumber) объекта «память» (тип TMemory). |
| Выход: | Объект число (тип T). |
| Постусловия: | Нет. |

***end*** TCMemory

# **Реализация**

В данной работе мы создали абстрактный тип данных “Память” и все необходимые для работы с ним операции, такие как: конструктор, запись, взять, добавить, очистить, считывание состояние памяти и считывания числа.

# **Демонстрация работы**



# **Вывод**

Мы научились работать в среде Visual Studio, а именно разрабатывать в ней модульные тесты для тестирования наших функции и классов на языке C#. Данная среда отлично подходит для выполнения модульного тестирования и автоматизации.

# **Список литературы**

1. Подбельский В.В., Фомин С.С.инт Курс программирования на языке Си: учебник. – М.:ДМК Пресс, 2012 – 384 с.
2. Павловская Т.А. C#. Программирование на языке высокого уровня: Учебник для вузов. - СПб. : Питер, 2014 - 432 с. : ил. - (Серия "Учебник для вузов").
3. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft.NET Framework 4 на языке C# . 3-е изд.: - СПб.:Питер, 2012 - 928 с. : ил.
4. Котляров, В. П. Основы тестирования программного обеспечения : учебное пособие для СПО / В. П. Котляров. — Саратов : Профобразование, 2019 — 335 c. — ISBN 978-5-4488-0364-2. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/86202.html (дата обращения: 21.08.2020). — Режим доступа: для авторизир. пользователей

# **Приложение**

## Листинг 1. TMemory.h

#pragma once

#include <string>

#include <iostream>

using namespace std;

template <class T>

class TMemory

{

private:

T FNumber;

bool FState;

public:

TMemory();

void write\_to\_memory(const T& e);

T get\_from\_memory();

void add\_in\_memory(const T& e);

void clear\_memory();

string get\_FState();

T get\_FNumber();

};

## Листинг 2. TMemory.cpp

//#include "pch.h"

#include "TMemory.h"

template<class T>

TMemory<T>::TMemory()

{

FNumber = T();

FState = false;

}

template<class T>

void TMemory<T>::write\_to\_memory(const T & e)

{

FNumber = e;

FState = true;

}

template<class T>

T TMemory<T>::get\_from\_memory()

{

return T(FNumber);

}

template<class T>

void TMemory<T>::add\_in\_memory(const T & e)

{

FNumber = FNumber + e;

FState = true;

}

template<class T>

void TMemory<T>::clear\_memory()

{

FNumber = T();

FState = false;

}

template<class T>

string TMemory<T>::get\_FState()

{

return FState ? "ON" : "OFF";

}

template<class T>

T TMemory<T>::get\_FNumber()

{

return FNumber;

}

## Листинг 3. TMemoryUnitTests.cpp

#include "pch.h"

#include "../lab11/TMemory.cpp"

#include "CppUnitTest.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace TMemoryUnitTests

{

TEST\_CLASS(TMemoryUnitTests)

{

public:

TEST\_METHOD(TMemory\_int\_constructor\_int\_OFF)

{

TMemory<int> test = TMemory<int>();

int expected = int();

int actual = test.get\_FNumber();

string expected\_state = "OFF";

string actual\_state = test.get\_FState();

Assert::AreEqual(expected, actual);

Assert::AreEqual(expected\_state, actual\_state);

}

TEST\_METHOD(TMemory\_float\_constructor\_float\_OFF)

{

TMemory<float> test = TMemory<float>();

float expected = float();

float actual = test.get\_FNumber();

string expected\_state = "OFF";

string actual\_state = test.get\_FState();

Assert::AreEqual(expected, actual);

Assert::AreEqual(expected\_state, actual\_state);

}

TEST\_METHOD(TMemory\_int\_write\_to\_memory\_12\_ON)

{

TMemory<int> test = TMemory<int>();

test.write\_to\_memory((string)"12+i\*3");

string expected = "12+i\*3";

int actual = test.get\_FNumber();

string expected\_state = "ON";

string actual\_state = test.get\_FState();

Assert::AreEqual(expected, actual);

Assert::AreEqual(expected\_state, actual\_state);

}

TEST\_METHOD(TMemory\_float\_write\_to\_memory\_12\_55\_ON)

{

TMemory<float> test = TMemory<float>();

test.write\_to\_memory(12.55);

float expected = 12.55;

float actual = test.get\_FNumber();

string expected\_state = "ON";

string actual\_state = test.get\_FState();

Assert::AreEqual(expected, actual);

Assert::AreEqual(expected\_state, actual\_state);

}

TEST\_METHOD(TMemory\_int\_get\_from\_memory\_12)

{

TMemory<int> test = TMemory<int>();

test.write\_to\_memory(12);

int expected = 12;

int actual = test.get\_from\_memory();

Assert::AreEqual(expected, actual);

}

TEST\_METHOD(TMemory\_float\_get\_from\_memory\_12\_55)

{

TMemory<float> test = TMemory<float>();

test.write\_to\_memory(12.55);

float expected = 12.55;

float actual = test.get\_from\_memory();

Assert::AreEqual(expected, actual);

}

TEST\_METHOD(TMemory\_int\_add\_in\_memory\_12\_12\_24)

{

TMemory<int> test = TMemory<int>();

test.write\_to\_memory(12);

test.add\_in\_memory(12);

int expected = 24;

int actual = test.get\_from\_memory();

Assert::AreEqual(expected, actual);

}

TEST\_METHOD(TMemory\_float\_add\_in\_memory\_12\_55\_12\_55\_25\_1)

{

TMemory<float> test = TMemory<float>();

test.write\_to\_memory(12.55);

test.add\_in\_memory(12.55);

float expected = 25.1;

float actual = test.get\_from\_memory();

Assert::AreEqual(expected, actual);

}

TEST\_METHOD(TMemory\_int\_clear\_memory)

{

TMemory<int> test = TMemory<int>();

test.write\_to\_memory(12);

test.clear\_memory();

int expected = 0;

int actual = test.get\_from\_memory();

string expected\_state = "OFF";

string actual\_state = test.get\_FState();

Assert::AreEqual(expected, actual);

Assert::AreEqual(expected\_state, actual\_state);

}

TEST\_METHOD(TMemory\_float\_clear\_memory)

{

TMemory<float> test = TMemory<float>();

test.write\_to\_memory(12.55);

test.clear\_memory();

float expected = 0;

float actual = test.get\_from\_memory();

string expected\_state = "OFF";

string actual\_state = test.get\_FState();

Assert::AreEqual(expected, actual);

Assert::AreEqual(expected\_state, actual\_state);

}

};

}