Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Сибирский государственный университет

телекоммуникаций и информатики» (СибГУТИ)

Кафедра прикладной математики и кибернетики

Лабораторная работа

«Шаблон класса память на одно число»

Выполнил:

Студент группы ИП-113

Шпилев Д. И.

Работу проверил:

старший преподаватель кафедры ПМиК

Агалаков А.А.

Новосибирск 2024 г.

**Содержание**

[**1.** **Задание** 3](#_Toc177578058)

[**2.** **Исходный код программы** 4](#_Toc177578059)

[**2.1.** **Код программы** 4](#_Toc177578060)

[**2.2** **Код тестов** 9](#_Toc177578061)

[**3.** **Результаты модульных тестов** 11](#_Toc177578062)

[**4.** **Вывод** 11](#_Toc177578063)

1. **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).

1. **Исходный код программы**
   1. **Код программы**

**UMemory.h**

#pragma once

#include "UANumber.h"

template <class T>

class TMemory

{

public:

enum State {

On,

Off

};

TMemory();

TMemory(T number);

TMemory(std::shared\_ptr<TANumber> number);

State getState() { return m\_state; };

void write(T number);

void write(std::shared\_ptr<TANumber> number);

std::shared\_ptr<TANumber> take();

void add(std::shared\_ptr<TANumber> B);

void clear();

private:

State m\_state;

std::shared\_ptr<TANumber> m\_number;

};

template<class T>

inline TMemory<T>::TMemory()

{

static\_assert(std::is\_base\_of<TANumber, T>::value, "T must inherit from TANumber");

m\_number = std::make\_shared<T>();

m\_state = Off;

}

template<class T>

inline TMemory<T>::TMemory(T number)

{

static\_assert(std::is\_base\_of<TANumber, T>::value, "T must inherit from TANumber");

m\_number = std::make\_shared<T>(number);

m\_state = Off;

}

template<class T>

inline TMemory<T>::TMemory(std::shared\_ptr<TANumber> number)

{

m\_number = number;

m\_state = Off;

}

template<class T>

inline void TMemory<T>::write(T number)

{

static\_assert(std::is\_base\_of<TANumber, T>::value, "T must inherit from TANumber");

m\_number = std::make\_shared<T>(number);

m\_state = On;

}

template<class T>

inline void TMemory<T>::write(std::shared\_ptr<TANumber> number)

{

m\_number = number;

m\_state = On;

}

template<class T>

inline std::shared\_ptr<TANumber> TMemory<T>::take()

{

m\_state = On;

return m\_number;

}

template<class T>

inline void TMemory<T>::add(std::shared\_ptr<TANumber> B)

{

m\_number = \*m\_number + \*B;

}

template<class T>

inline void TMemory<T>::clear()

{

m\_number = std::make\_shared<T>();

m\_state = Off;

}

TEST\_CLASS(MemoryTest)

{

public:

TEST\_METHOD(TestWriteAndTake)

{

TMemory<TComplex> memory;

TComplex complexNumber(5, 13);

memory.write(complexNumber);

auto result = memory.take();

TComplex\* resultComplex = dynamic\_cast<TComplex\*>(result.get());

Assert::IsNotNull(resultComplex);

Assert::AreEqual(5.0, resultComplex->getActual());

Assert::AreEqual(13.0, resultComplex->getImaginary());

}

TEST\_METHOD(TestAdd)

{

TMemory<TComplex> memory;

auto complexNumber1 = std::make\_shared<TComplex>(3, 4);

auto complexNumber2 = std::make\_shared<TComplex>(1, 2);

memory.write(complexNumber1);

memory.add(complexNumber2);

auto result = memory.take();

TComplex\* resultComplex = dynamic\_cast<TComplex\*>(result.get());

Assert::IsNotNull(resultComplex);

Assert::AreEqual(4.0, resultComplex->getActual());

Assert::AreEqual(6.0, resultComplex->getImaginary());

}

TEST\_METHOD(TestClear)

{

TMemory<TComplex> memory;

TComplex complexNumber(5, 13);

memory.write(complexNumber);

memory.clear();

auto result = memory.take();

TComplex\* resultComplex = dynamic\_cast<TComplex\*>(result.get());

Assert::IsNotNull(resultComplex);

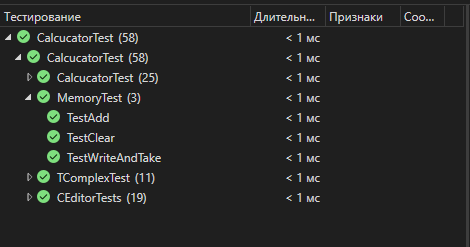
Assert::AreEqual(0.0, resultComplex->getActual());

Assert::AreEqual(0.0, resultComplex->getImaginary());

}

};

1. **Результаты модульных тестов**

****

1. **Вывод**

По итогам данной лабораторной работе были сформированы практические навыки реализации абстрактных типов данных в соответствии с заданной спецификацией с помощью классов С++ и их модульного тестирования.