

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
2. Исходный код программы	3
2.1. Код программы	3
3. Результаты модульных тестов	6
4. Вывод	6

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 C++.
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).

2. Исходный код программы

2.1. 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());
    }
};

```

3. Результаты модульных тестов

Тестирование	Длительн...	Признаки	Соо...
▲ ✓ CalcucatorTest (58)	< 1 мс		
▲ ✓ CalcucatorTest (58)	< 1 мс		
▷ ✓ CalcucatorTest (25)	< 1 мс		
▲ ✓ MemoryTest (3)	< 1 мс		
✓ TestAdd	< 1 мс		
✓ TestClear	< 1 мс		
✓ TestWriteAndTake	< 1 мс		
▷ ✓ TComplexTest (11)	< 1 мс		
▷ ✓ CEditorTests (19)	< 1 мс		

4. Вывод

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