

Федеральное агентство связи  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Сибирский государственный университет  
телекоммуникаций и информатики»

## **Лабораторная работа №7**

Выполнил: студент 4 курса

ИВТ, гр. ИП-713

Михеев Н.А.

Проверил: ассистент кафедры

ПМиК

Агалаков А.А.

Новосибирск, 2020 г.

## Цель

Сформировать практические навыки реализации параметризованного абстрактного типа данных с помощью шаблона классов C++.

## 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.

## Реализация и описание

Абстрактный тип данных «память» был реализован посредством создания шаблона класса. Шаблон класса имеет два поля, которые хранят в себе число (объект класса T) и состояние памяти. Также были реализованы процедуры для взаимодействия с объектами данного класса.

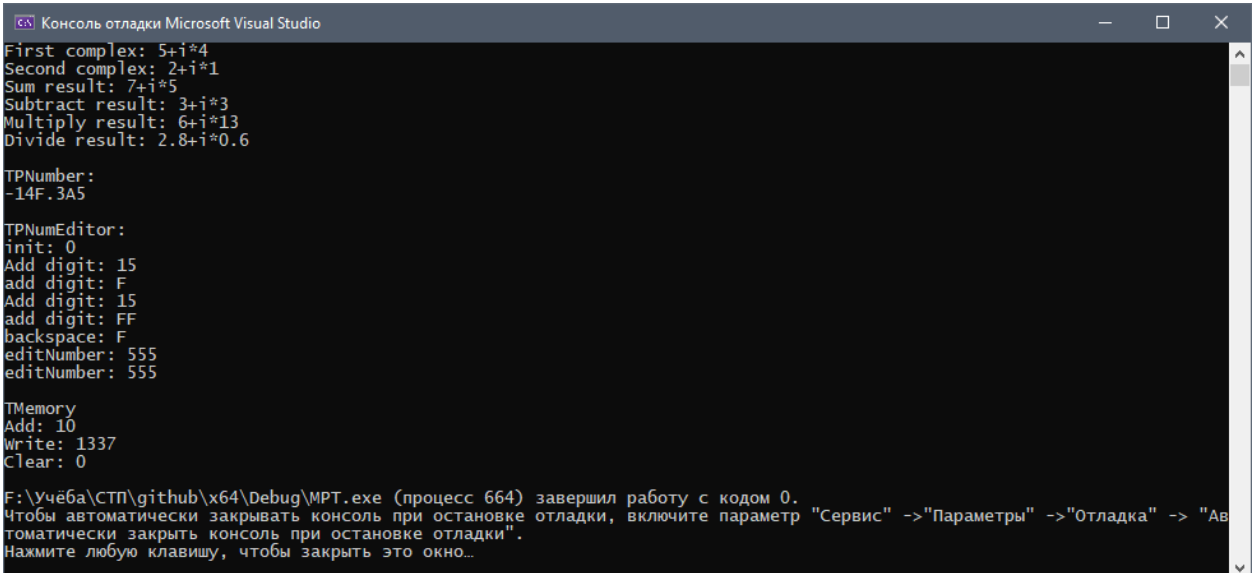
- TMemory<T>::TMemory(T FNumber) – конструктор инициализирует поле FNumber объекта «память» (тип TMemory) объектом «число» (тип T) со значением по умолчанию.
- void TMemory<T>::write(T FNumber) – метод, который записывает T FNumber в поле FNumber.
- T TMemory<T>::get() – Создает и возвращает копию объекта хранящегося в объекте «память» (тип TMemory) в поле FNumber.
- void TMemory<T>::add(T FNumber) – в поле FNumber объекта «память» (тип TMemory) записывается объект типа T, полученный в результате сложения числа (E) и числа, хранящегося в памяти в поле FNumber.

- `void TMemory<T>::clear()` – 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).
- `string TMemory<T>::readFState()` – Копирует и возвращает значение поля FState «состояние памяти» объекта «память» (тип TMemory) в формате строки.
- `T TMemory<T>::readNumber()` – Копирует и возвращает значение поля «число» (FNumber) объекта «память» (тип TMemory).

## Заключение

В ходе данной работы согласно спецификациям задания был реализован абстрактный тип данных «память». Также был получен практический опыт написания шаблонных функций на языке программирования C++.

## Скриншоты



```

Консоль отладки Microsoft Visual Studio
First complex: 5+i*4
Second complex: 2+i*1
Sum result: 7+i*5
Subtract result: 3+i*3
Multiply result: 6+i*13
Divide result: 2.8+i*0.6

TPNumber:
-14F.3A5

TPNumEditor:
init: 0
Add digit: 15
add digit: F
Add digit: 15
add digit: FF
backspace: F
editNumber: 555
editNumber: 555

TMemory
Add: 10
Write: 1337
Clear: 0

F:\Учёба\СТП\github\x64\Debug\MPT.exe (процесс 664) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...

```

Рис. 1 – пример работы программы

Обозреватель тестов			
<div> <div>▶▶▶▶</div> <div>185</div> <div>185</div> <div>0</div> <div> <div>🔍</div> <div>Поиск в обозревателе тестов</div> <div>🔍</div> </div> </div>			
Тестирование	Длительн...	Признаки	Сообщение об ошибке
<div> <div>▶</div> <div>✓ MPT_Test (185)</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ MPT_Tests (185)</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ TComplexTest (35)</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ TFracEditorTest (24)</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ TFracTest (35)</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ TMemory_Test (20)</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ ADD_1</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ ADD_2</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ ADD_3</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ CLEAR_1</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ CONSTR_1</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ CONSTR_2</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ CONSTR_3</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ CONSTR_4</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ GET_1</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ GET_2</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ GET_3</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ READ_FSTATE_1</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ READ_FSTATE_2</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ READ_NUMBER_1</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ READ_NUMBER_2</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ READ_NUMBER_3</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ READ_NUMBER_4</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ WRITE_1</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ WRITE_2</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ WRITE_3</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ TPNumEditorTest (17)</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ TPNumberTest (29)</div> <div>&lt; 1 мс</div> </div>	< 1 мс		
<div> <div>▶</div> <div>✓ TProc_Test (25)</div> <div>&lt; 1 мс</div> </div>	< 1 мс		

Рис. 2 – сводка проведённого тестирования программы

## Код программы

### TMemory.h

```

#pragma once
#include <string>

using namespace std;

enum StateOfMemory {
    _Off,
    _On
};

template <class T>
class TMemory
{
private:
    T FNumber;
    int FState;
public:
    TMemory(T FNumber);

```

```

        void write(T FNumber);
        T get();
        void add(T FNumber);
        void clear();
        string readFState();
        T readNumber();
        ~TMemory();
};

template <class T>
TMemory<T>::TMemory(T FNumber)
{
    string type = typeid(T).name();
    if (type != "class TFracEditor" &&
        type != "class TPNumber" &&
        type != "class TComplexEditor" &&
        type != "bool" &&
        type != typeid(string).name())
    {
        this->FNumber = FNumber;
        this->FState = _Off;
    }
    else {
        throw invalid_argument("Invalid type");
    }
}

template<class T>
void TMemory<T>::write(T FNumber)
{
    this->FNumber = FNumber;
    this->FState = _On;
}

template<class T>
T TMemory<T>::get()
{
    this->FState = _On;
    return T(this->FNumber);
}

template<class T>
void TMemory<T>::add(T FNumber)
{
    this->FNumber = this->FNumber + FNumber;
    this->FState = _On;
}

template<class T>
void TMemory<T>::clear()
{
    T newObj;
    this->FNumber = newObj;
    this->FState = _Off;
}

template<class T>
string TMemory<T>::readFState()
{
    return string(to_string(this->FState));
}

template<class T>
T TMemory<T>::readNumber()
{
    return this->FNumber;
}

template<class T>
TMemory<T>::~TMemory() {}

```

## TMemory\_Test.cpp

```

#include "pch.h"
#include "CppUnitTest.h"

```

```

#include "../MPT/TMemory.h"
#include "../MPT/TFrac.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MPT_Tests
{
    TEST_CLASS(TMemory_Test)
    {
    public:
        TEST_METHOD(ADD_1)
        {
            TFrac tfrac(1, 5);
            TMemory<TFrac> tmem(tfrac);
            tmem.add(tfrac);
            Assert::AreEqual(string("2/5"), tmem.get().getFractionString());
        }

        TEST_METHOD(ADD_2)
        {
            TFrac tfrac(1, 4);
            TMemory<TFrac> tmem(tfrac);
            tmem.add(tfrac);
            Assert::AreEqual(string("1/2"), tmem.get().getFractionString());
        }

        TEST_METHOD(ADD_3)
        {
            TFrac tfrac(1, 999);
            TMemory<TFrac> tmem(tfrac);
            tmem.add(tfrac);
            Assert::AreEqual(string("2/999"), tmem.get().getFractionString());
        }

        TEST_METHOD(WRITE_1)
        {
            TFrac tfrac(1, 5);
            TFrac tfrac2(5, 7);
            TMemory<TFrac> tmem(tfrac);
            tmem.write(tfrac2);
            Assert::AreEqual(string("5/7"), tmem.get().getFractionString());
        }

        TEST_METHOD(WRITE_2)
        {
            TFrac tfrac(1, 5);
            TFrac tfrac2(1, 10);
            TMemory<TFrac> tmem(tfrac);
            tmem.write(tfrac2);
            Assert::AreEqual(string("1/10"), tmem.get().getFractionString());
        }

        TEST_METHOD(WRITE_3)
        {
            TFrac tfrac(1, 5);
            TFrac tfrac2(2, 99);
            TMemory<TFrac> tmem(tfrac);
            tmem.write(tfrac2);
            Assert::AreEqual(string("2/99"), tmem.get().getFractionString());
        }

        TEST_METHOD(GET_1)
        {
            TFrac tfrac(1, 5);
            TMemory<TFrac> tmem(tfrac);
            Assert::AreEqual(string("1/5"), tmem.get().getFractionString());
        }

        TEST_METHOD(GET_2)
        {
            TFrac tfrac(2, 5);
            TMemory<TFrac> tmem(tfrac);
            Assert::AreEqual(string("2/5"), tmem.get().getFractionString());
        }

        TEST_METHOD(GET_3)

```

```

    {
        TFrac tfrac(3, 5);
        TMemory<TFrac> tmem(tfrac);
        Assert::AreEqual(string("3/5"), tmem.get().getFractionString());
    }

    TEST_METHOD(CLEAR_1)
    {
        TFrac tfrac(1, 5);
        TMemory<TFrac> tmem(tfrac);
        tmem.clear();
        Assert::AreEqual(string("0/1"), tmem.get().getFractionString());
    }

    TEST_METHOD(READ_FSTATE_1)
    {
        TFrac tfrac(1, 5);
        TMemory<TFrac> tmem(tfrac);
        tmem.clear();
        Assert::AreEqual(string("0"), tmem.readFState());
    }

    TEST_METHOD(READ_FSTATE_2)
    {
        TFrac tfrac(1, 5);
        TMemory<TFrac> tmem(tfrac);
        tmem.add(tfrac);
        Assert::AreEqual(string("1"), tmem.readFState());
    }

    TEST_METHOD(READ_NUMBER_1)
    {
        TFrac tfrac(1, 5);
        TMemory<TFrac> tmem(tfrac);
        Assert::AreEqual(tfrac.getFractionString(),
tmem.readNumber().getFractionString());
    }

    TEST_METHOD(READ_NUMBER_2)
    {
        TFrac tfrac(2, 5);
        TMemory<TFrac> tmem(tfrac);
        Assert::AreEqual(tfrac.getFractionString(),
tmem.readNumber().getFractionString());
    }

    TEST_METHOD(READ_NUMBER_3)
    {
        TFrac tfrac(4, 5);
        TMemory<TFrac> tmem(tfrac);
        Assert::AreEqual(tfrac.getFractionString(),
tmem.readNumber().getFractionString());
    }

    TEST_METHOD(READ_NUMBER_4)
    {
        TFrac tfrac(5, 5);
        TMemory<TFrac> tmem(tfrac);
        Assert::AreEqual(tfrac.getFractionString(),
tmem.readNumber().getFractionString());
    }

    };
}

```