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

Лабораторная работа №8

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 «Процессор», 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 «процессор» был реализован посредством создания шаблона класса. Шаблон класса имеет три поля, которые хранят в себе число (объект класса T), операцию и функцию. Также были реализованы процедуры для взаимодействия с объектами данного класса.

- TProc(const T& leftObj, const T& rightObj) – конструктор инициализирует поле FNumber объекта «память» (тип TMemory) объектом «число» (тип T) со значением по умолчанию.
- void resetProc() – метод, выполняющий сброс объекта TProc. Поле типа T инициализируется значением по умолчанию, процессор устанавливается в состояние: ‘операция не установлена’.
- void resetOper() – процессор устанавливается в состояние: ‘операция не установлена’.
- void doOper() – вызывает выполнение текущей операции (записанной в поле Operation). Операция (Operation) выполняется над значениями, хранящимися в полях Rop и Lop_Res. Результат сохраняется в поле

Lop_Res. Если Operation = None, никакие действия не выполняются. Состояние объекта не изменяется.

- void doFunc(TFunc func) – вызывает выполнение текущей функции (Func). Функция (Func) выполняется над значением, хранящимся в поле Rop. Результат сохраняется в нём же. Состояние объекта не изменяется.
- T readLeft() – создаёт и возвращает копию объекта, который хранится в поле Lop_Res.
- T readRight() – создаёт и возвращает копию объекта, который хранится в Rop.
- void writeLeft(const T& someObj) – создаёт копию объекта Operand и заносит её в поле Lop_Res.
- void writeRight(const T& someObj) – создаёт копию объекта Operand и заносит её в поле Rop.
- void writeOper(TOperations someOper) – создаёт копию объекта Operand и заносит её в поле Rop.
- TOperations readOper() – копирует и возвращает значение поля Operation.
- ~TProc() – деструктор для объектов класса TProc.

Заключение

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

Скриншоты

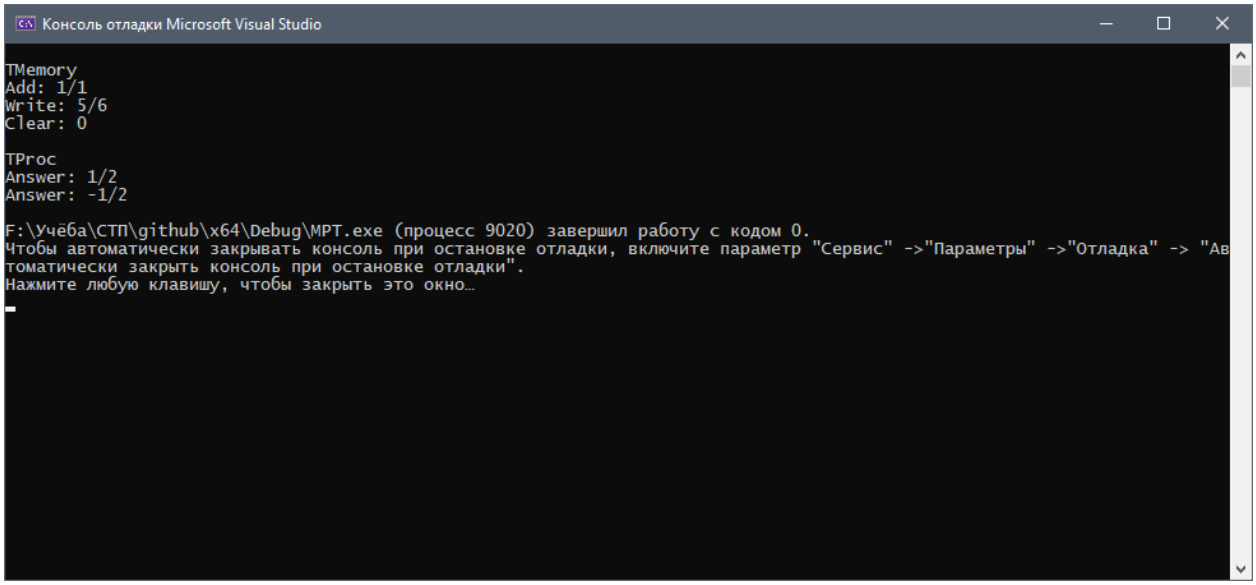

Рис. 1 – пример работы программы

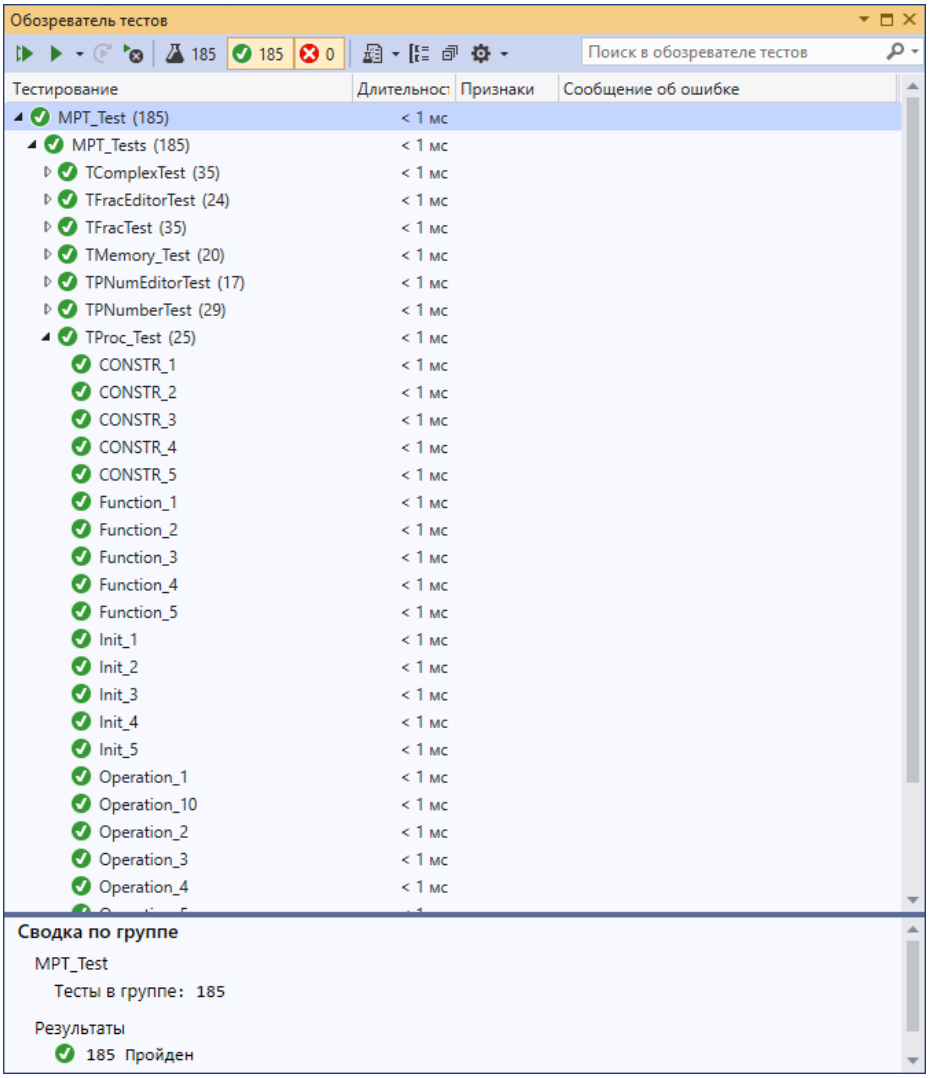

Рис. 2 – сводка проведённого тестирования программы

Код программы

TProc.h

```
#pragma once

enum TOperations
{
	None, Add, Sub, Mul, Div
};

enum TFunc
{
	Rev, Sqr
};

template <typename T = int>
class TProc
{
private:
	T Lop_Res;
	T Rop;
	TOperations operation;
public:
	TProc(const T& leftObj, const T& rightObj);
	void resetProc();
	void resetOper();
	void doOper();
	void doFunc(TFunc func);
	T readLeft();
	T readRight();
	void writeLeft(const T& someObj);
	void writeRight(const T& someObj);
	void writeOper(TOperations someOper);
	TOperations readOper();
	~TProc();
};

template<typename T>
TProc<T>::TProc(const T& leftObj, const T& rightObj)
{
	string type = typeid(T).name();
	if (type != "class TFracEditor" &&
		type != "class TPNumber" &&
		type != "class TComplexEditor" &&
		type != "bool" &&
		type != typeid(string).name())
	{
		operation = None;
		this->Lop_Res = leftObj;
		this->Rop = rightObj;
	}
	else {
		throw invalid_argument("Invalid type");
	}
}

template<typename T>
void TProc<T>::resetProc()
{
	operation = None;
	T clearObj;
	this->Lop_Res = this->Rop = clearObj;
}

template<typename T>
void TProc<T>::resetOper()
{
	operation = None;
}

template<typename T>
void TProc<T>::doOper()
{
	switch (operation) {
```

```
        case Add:
                Lop_Res = Lop_Res + Rop;
                break;
        case Sub:
                Lop_Res = Lop_Res - Rop;
                break;
        case Mul:
                Lop_Res = Lop_Res * Rop;
                break;
        case Div:
                Lop_Res = Lop_Res / Rop;
                break;
        default:
                break;
        }
}

template<typename T>
void TProc<T>::doFunc(TFunc func)
{
        switch (func) {
        case Rev:
                Rop = Rop.reverse();
                break;
        case Sqr:
                Rop = Rop * Rop;
                break;
        default:
                break;
        }
}

template<typename T>
T TProc<T>::readLeft()
{
        return T(this->Lop_Res);
}

template<typename T>
T TProc<T>::readRight()
{
        return T(this->Rop);
}

template<typename T>
void TProc<T>::writeLeft(const T& someObj)
{
        this->Lop_Res = someObj;
}

template<typename T>
void TProc<T>::writeRight(const T& someObj)
{
        this->Rop = someObj;
}

template<typename T>
void TProc<T>::writeOper(TOperations someOper)
{
        this->operation = someOper;
}

template<typename T>
TOperations TProc<T>::readOper()
{
        return operation;
}

template<typename T>
TProc<T>::~TProc() {}
```

TProc_Test.cpp

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

#include "../MPT/TProc.h"
```

```
#include "../MPT/TFrac.h"
#include "../MPT/TFracEditor.h"
#include "../MPT/TPNumber.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MPT_Tests
{
	TEST_CLASS(TProc_Test)
	{
	public:

		TEST_METHOD(CONSTR_1)
		{
			auto constr = [] {TFracEditor a; TProc b(a, a); };
			Assert::ExpectException<std::invalid_argument>(constr);
		}
		TEST_METHOD(CONSTR_2)
		{
			auto constr = [] {TPNumber a; TProc b(a, a); };
			Assert::ExpectException<std::invalid_argument>(constr);
		}
		TEST_METHOD(CONSTR_3)
		{
			auto constr = [] {bool a; TProc b(a, a); };
			Assert::ExpectException<std::invalid_argument>(constr);
		}
		TEST_METHOD(CONSTR_4)
		{
			auto constr = [] {string a; TProc b(a, a); };
			Assert::ExpectException<std::invalid_argument>(constr);
		}
		TEST_METHOD(CONSTR_5)
		{
			auto constr = [] {string a("abc"); TProc b(a, a); };
			Assert::ExpectException<std::invalid_argument>(constr);
		}
		TEST_METHOD(Init_1) { // 0/1 0/1
			TFrac leftFrac;
			TFrac rightFrac;
			TProc<TFrac> obj(leftFrac, rightFrac);
			TFrac answer;
			Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
		}
		TEST_METHOD(Init_2) { // 11/3 11/3
			TFrac leftFrac(11, 3);
			TFrac rightFrac;
			TProc<TFrac> obj(leftFrac, rightFrac);
			TFrac answer(11, 3);
			Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
		}
		TEST_METHOD(Init_3) { // 17/9 17/9
			TFrac leftFrac(16, 4);
			TFrac rightFrac(17, 9);
			TProc<TFrac> obj(leftFrac, rightFrac);
			TFrac answer(17, 9);
			Assert::AreEqual(answer.getFractionString(), obj.readRight().getFractionString());
		}

		TEST_METHOD(Init_4) { // 17/9 17/9
			TFrac leftFrac(500, 4);
			TFrac rightFrac(-300, 9);
			TProc<TFrac> obj(leftFrac, rightFrac);
			TFrac answer(-300, 9);
			Assert::AreEqual(answer.getFractionString(), obj.readRight().getFractionString());
		}

		TEST_METHOD(Init_5) { // 17/9 17/9
			TFrac leftFrac(500, 4);
			TFrac rightFrac(123, 45);
			TProc<TFrac> obj(leftFrac, rightFrac);
			TFrac answer(123, 45);
			Assert::AreEqual(answer.getFractionString(), obj.readRight().getFractionString());
		}

		TEST_METHOD(Operation_1) { // 1/2 + 1/2 = 1/1
			TFrac leftFrac(1, 2);
```

```
        TFrac rightFrac(1, 2);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TOperations oper = Add;
        obj.writeOper(oper);
        obj.doOper();
        TFrac answer(1, 1);
        Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
}

TEST_METHOD(Operation_2) { // 3/4 - 5/6 = -1/12
        TFrac leftFrac(3, 4);
        TFrac rightFrac(5, 6);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TOperations oper = Sub;
        obj.writeOper(oper);
        obj.doOper();
        TFrac answer(-1, 12);
        Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
}
TEST_METHOD(Operation_3) { // 12/7 * 5/9 = 20/21
        TFrac leftFrac(12, 7);
        TFrac rightFrac(5, 9);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TOperations oper = Mul;
        obj.writeOper(oper);
        obj.doOper();
        TFrac answer(20, 21);
        Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
}
TEST_METHOD(Operation_4) { // 56/7 : -22/3 = -12/11
        TFrac leftFrac(56, 7);
        TFrac rightFrac(-22, 3);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TOperations oper = Div;
        obj.writeOper(oper);
        obj.doOper();
        TFrac answer(-12, 11);
        Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
}

TEST_METHOD(Operation_5) {
        TFrac leftFrac(15, 10);
        TFrac rightFrac(55, 60);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TOperations oper = Add;
        obj.writeOper(oper);
        obj.doOper();
        TFrac answer(29, 12);
        Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
}

TEST_METHOD(Operation_6) {
        TFrac leftFrac(1, 1);
        TFrac rightFrac(1, 1);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TOperations oper = Add;
        obj.writeOper(oper);
        obj.doOper();
        TFrac answer(2, 1);
        Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
}

TEST_METHOD(Operation_7) {
        TFrac leftFrac(3, 1);
        TFrac rightFrac(1, 1);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TOperations oper = Add;
        obj.writeOper(oper);
        obj.doOper();
        TFrac answer(4, 1);
        Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
}

TEST_METHOD(Operation_8) { // 12/7 * 5/9 = 20/21
        TFrac leftFrac(7, 7);
        TFrac rightFrac(7, 7);
        TProc<TFrac> obj(leftFrac, rightFrac);
```

```
        TOperations oper = Mul;
        obj.writeOper(oper);
        obj.doOper();
        TFrac answer(1, 1);
        Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
}

TEST_METHOD(Operation_9) { // 12/7 * 5/9 = 20/21
        TFrac leftFrac(8, 7);
        TFrac rightFrac(8, 7);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TOperations oper = Mul;
        obj.writeOper(oper);
        obj.doOper();
        TFrac answer(64, 49);
        Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
}

TEST_METHOD(Operation_10) { // 12/7 * 5/9 = 20/21
        TFrac leftFrac(5, 7);
        TFrac rightFrac(1, 1);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TOperations oper = Mul;
        obj.writeOper(oper);
        obj.doOper();
        TFrac answer(5, 7);
        Assert::AreEqual(answer.getFractionString(), obj.readLeft().getFractionString());
}
TEST_METHOD(Function_1) { // reverse
        TFrac leftFrac(56, 7);
        TFrac rightFrac(-22, 3);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TFunc funcOper = Rev;
        obj.doFunc(funcOper);
        TFrac answer(-3, 22);
        Assert::AreEqual(answer.getFractionString(), obj.readRight().getFractionString());
}
TEST_METHOD(Function_2) {
        TFrac leftFrac;
        TFrac rightFrac(-22, 3);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TFunc funcOper = Sqr;
        obj.doFunc(funcOper);
        TFrac answer(484, 9);
        Assert::AreEqual(answer.getFractionString(), obj.readRight().getFractionString());
}

TEST_METHOD(Function_3) {
        TFrac leftFrac;
        TFrac rightFrac(1, 1);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TFunc funcOper = Sqr;
        obj.doFunc(funcOper);
        TFrac answer(1, 1);
        Assert::AreEqual(answer.getFractionString(), obj.readRight().getFractionString());
}

TEST_METHOD(Function_4) {
        TFrac leftFrac;
        TFrac rightFrac(2, 2);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TFunc funcOper = Sqr;
        obj.doFunc(funcOper);
        TFrac answer(1, 1);
        Assert::AreEqual(answer.getFractionString(), obj.readRight().getFractionString());
}

TEST_METHOD(Function_5) {
        TFrac leftFrac;
        TFrac rightFrac(3, 9);
        TProc<TFrac> obj(leftFrac, rightFrac);
        TFunc funcOper = Sqr;
        obj.doFunc(funcOper);
        TFrac answer(1, 9);
        Assert::AreEqual(answer.getFractionString(), obj.readRight().getFractionString());
}
```

```
        };
}
```