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

Лабораторная работа №3

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 абстрактных типов данных в соответствии с заданной спецификацией с помощью классов C++ и их модульного тестирования.

Задание

1. Реализовать абстрактный тип данных «р-ичное число», используя класс C++ в соответствии с приведенной ниже спецификацией.

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 “р-ичное число” был реализован посредством создания класса. Класс имеет три поля, которые хранят в себе само число, основание и точность представления числа. Также были реализованы несколько конструкторов (перегруженных) и процедуры для взаимодействия с объектами данного класса.

- TPNumber(double, int, int) – конструктор с инициализацией числа, основанием системы счисления, точности представления.
- TPNumber(string, string, string) - конструктор с инициализацией числа, основанием системы счисления, точности представления (получение значений из string).
- TPNumber copy() – метод для копирования объекта.
- TPNumber inverse() – метод для получения обратного числа.
- TPNumber square() – метод, реализующий возведение числа в квадрат.

- TPNumber add(TPNumber) – метод, реализующий сложение р-ичных чисел.
- TPNumber subtr(TPNumber) – метод, реализующий вычитание р-ичных чисел.
- TPNumber mult(TPNumber) – метод, реализующий умножение р-ичных чисел.
- TPNumber div(TPNumber) – метод, реализующий деление р-ичных чисел.
- double getNumber() – метод для получения числа из класса.
- string getNumberString() – метод для получения числа из класса в string.
- int getBase() – метод для получения основания системы счисления.
- string getBaseString() - метод для получения основания системы счисления в string.
- int getPrecision() - метод для получения точности представления.
- string getPrecisionString() - метод для получения точности представления в string.
- void setBase(int) - метод для изменения основания системы счисления.
- void setBase(string) – метод для изменения основания системы счисления.
- void setPrecision(int) – метод для изменения точности представления.
- void setPrecision(string) – метод для изменения точности представления.

Заключение

В ходе данной работы согласно спецификациям задания был реализован абстрактный тип данных «р-ичное число». Также был получен

практический опыт написания модульных тестов для тестирования каждой операции.

Скриншоты

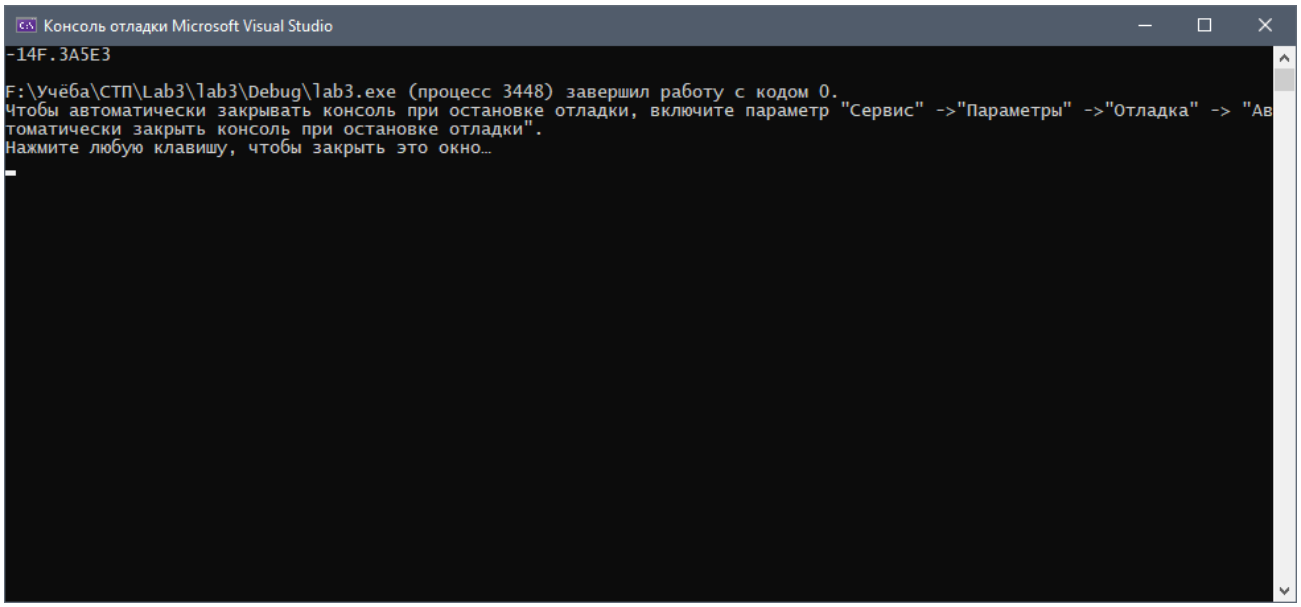

Рис. 1 – пример работы программы

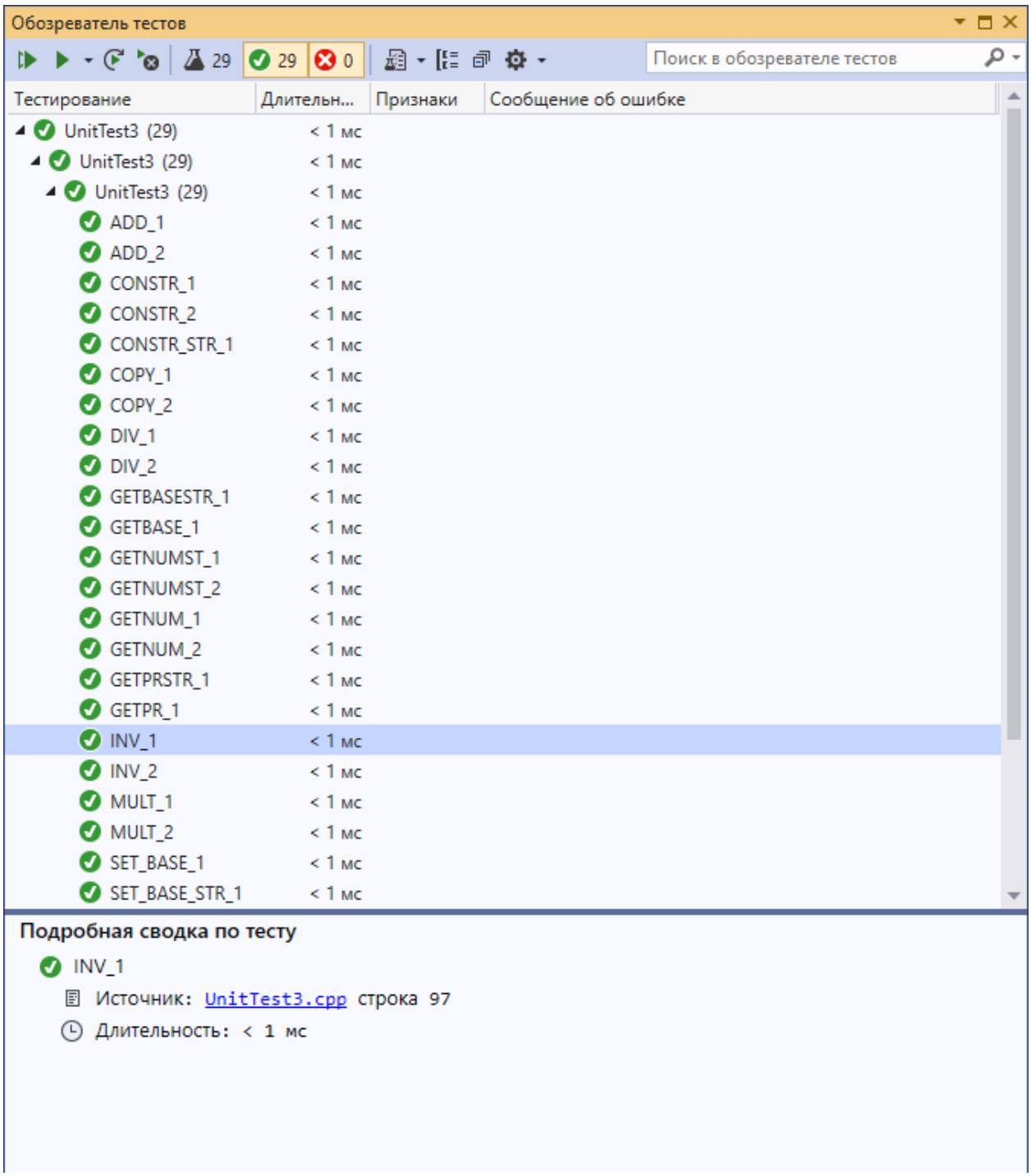

Рис. 2 – сводка проведённого тестирования программы

Код программы

TPNumber.h

```
#pragma once
#include <string>

using namespace std;

const string symbols = "0123456789ABCDEF";

class TPNumber
{
private:
    double number;
```

```
    int base, precision;
public:
    TPNumber(double, int, int);
    TPNumber(string, string, string);

    TPNumber copy();
    TPNumber inverse();
    TPNumber square();

    TPNumber add(TPNumber);
    TPNumber subtr(TPNumber);
    TPNumber mult(TPNumber);
    TPNumber div(TPNumber);

    double getNumber();
    string getNumberString();

    int getBase();
    string getBaseString();

    int getPrecision();
    string getPrecisionString();

    void setBase(int);
    void setBase(string);

    void setPrecision(int);
    void setPrecision(string);
};
```

TPNumber.cpp

```
#include "pch.h"
#include "TPNumber.h"

using namespace std;

TPNumber::TPNumber(double a, int b, int c) {
    if (b >= 2 && b <= 16 && c >= 0) {
        this->number = a;
        this->base = b;
        this->precision = c;
    }
    else {
        this->number = 0.0;
        this->base = 10;
        this->precision = 0;
    }
}

TPNumber::TPNumber(string a, string b, string c) {
    double numberTemp = stod(a);
    int baseTemp = stoi(b);
    int precisionTemp = stoi(c);

    if (baseTemp >= 2 && baseTemp <= 16 && precisionTemp >= 0) {
        this->number = numberTemp;
        this->base = baseTemp;
        this->precision = precisionTemp;
    }
    else {
        this->number = 0.0;
        this->base = 10;
        this->precision = 0;
    }
}

TPNumber TPNumber::copy() {
    return { number, base, precision };
}

TPNumber TPNumber::add(TPNumber right) {
    if (this->base == right.base && this->precision == right.precision) {
        return { this->number + right.number, this->base, this->precision };
    }
    else return { 0.0, 10, 0 };
}
```

```
TPNumber TPNumber::mult(TPNumber right) {
    if (this->base == right.base && this->precision == right.precision) {
        return { this->number * right.number, this->base, this->precision };
    }
    else return { 0.0, 10, 0 };
}

TPNumber TPNumber::subtr(TPNumber right) {
    if (this->base == right.base && this->precision == right.precision) {
        return { this->number - right.number, this->base, this->precision };
    }
    else return { 0.0, 10, 0 };
}

TPNumber TPNumber::div(TPNumber right) {
    if (this->base == right.base && this->precision == right.precision) {
        return { this->number / right.number, this->base, this->precision };
    }
    else return { 0.0, 10, 0 };
}

TPNumber TPNumber::inverse() {
    return { 1 / this->number, this->base, this->precision };
}

TPNumber TPNumber::square() {
    return { this->number * this->number, this->base, this->precision };
}

double TPNumber::getNumber() {
    return this->number;
}

string TPNumber::getNumberString() {
    double tempNumber = fabs(this->number);
    int leftPart = (int)floor(tempNumber); // Целая часть числа
    double rightPart = tempNumber - floor(tempNumber); // Дробная часть числа
    string result;

    while (leftPart > 0) {
        int ost = leftPart % this->base;
        result.insert(0, 1, symbols[ost]);
        leftPart /= this->base;
    }

    if (result.empty()) result = "0";

    if (rightPart == 0.0) return result;
    else {
        result.append(1, '.');
        for (int i = 1; i <= precision; i++) {
            rightPart *= base;
            result.append(1, symbols[(int)floor(rightPart)]);
            rightPart -= floor(rightPart);
            if (rightPart == 0.0) break;
        }
    }
    if (this->number < 0) result = result = "-" + result;

    return result;
}

int TPNumber::getBase() {
    return this->base;
}

string TPNumber::getBaseString() {
    return to_string(this->base);
}

int TPNumber::getPrecision() {
    return this->precision;
}

string TPNumber::getPrecisionString() {
    return to_string(this->precision);
```

```
}

void TPNumber::setBase(int newBase) {
    if (newBase >= 2 && newBase <= 16) this->base = newBase;
}

void TPNumber::setBase(string stringBase) {
    int newBase = stoi(stringBase);
    if (newBase >= 2 && newBase <= 16) this->base = newBase;
}

void TPNumber::setPrecision(int newPrecision) {
    if (newPrecision >= 0) this->precision = newPrecision;
}

void TPNumber::setPrecision(string stringPrecision) {
    int newPrecision = stoi(stringPrecision);
    if (newPrecision >= 0) this->precision = newPrecision;
}
```

UnitTest3.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../lab3/TPNumber.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest3
{
	TEST_CLASS(UnitTest3)
	{
	public:
		TEST_METHOD(CONSTR_1)
		{
			Assert::AreEqual(10, TPNumber(1, 10, 2).getBase());
		}

		TEST_METHOD(CONSTR_2)
		{
			Assert::AreEqual(10, TPNumber(1, -1, 2).getBase());
		}

		TEST_METHOD(CONSTR_STR_1)
		{
			Assert::AreEqual(10, TPNumber("1", "10", "2").getBase());
		}

		TEST_METHOD(COPY_1)
		{
			TPNumber a(1, 10, 2);
			TPNumber b = a.copy();
			Assert::AreEqual(1.0, b.getNumber());
		}

		TEST_METHOD(COPY_2)
		{
			TPNumber a(-1.0, 10, 2);
			TPNumber b = a.copy();
			Assert::AreEqual(-1.0, b.getNumber());
		}

		TEST_METHOD(ADD_1)
		{
			TPNumber a(5.0, 10, 2);
			TPNumber b(5.0, 10, 2);
			Assert::AreEqual(10.0, a.add(b).getNumber());
		}

		TEST_METHOD(ADD_2)
		{
			TPNumber a(-5.0, 10, 2);
			TPNumber b(-5.0, 10, 2);
			Assert::AreEqual(-10.0, a.add(b).getNumber());
		}

		TEST_METHOD(MULT_1)
```

```
{
        TPNumber a(5.0, 10, 2);
        TPNumber b(5.0, 10, 2);
        Assert::AreEqual(25.0, a.mult(b).getNumber());
}

TEST_METHOD(MULT_2)
{
        TPNumber a(-5.0, 10, 2);
        TPNumber b(-5.0, 10, 2);
        Assert::AreEqual(25.0, a.mult(b).getNumber());
}

TEST_METHOD(SUBTR_1)
{
        TPNumber a(5.0, 10, 2);
        TPNumber b(5.0, 10, 2);
        Assert::AreEqual(0.0, a.subtr(b).getNumber());
}

TEST_METHOD(SUBTR_2)
{
        TPNumber a(5.0, 10, 2);
        TPNumber b(-5.0, 10, 2);
        Assert::AreEqual(10.0, a.subtr(b).getNumber());
}

TEST_METHOD(DIV_1)
{
        TPNumber a(5.0, 10, 2);
        TPNumber b(5.0, 10, 2);
        Assert::AreEqual(1.0, a.div(b).getNumber());
}

TEST_METHOD(DIV_2)
{
        TPNumber a(5.0, 10, 2);
        TPNumber b(-5.0, 10, 2);
        Assert::AreEqual(-1.0, a.div(b).getNumber());
}

TEST_METHOD(INV_1)
{
        TPNumber a(5.0, 10, 3);
        Assert::AreEqual(0.2, a.inverse().getNumber());
}

TEST_METHOD(INV_2)
{
        TPNumber a(1.0, 10, 2);
        Assert::AreEqual(1.0, a.inverse().getNumber());
}

TEST_METHOD(SQ_1)
{
        TPNumber a(2.0, 10, 2);
        Assert::AreEqual(4.0, a.square().getNumber());
}

TEST_METHOD(SQ_2)
{
        TPNumber a(1.0, 10, 2);
        Assert::AreEqual(1.0, a.square().getNumber());
}

TEST_METHOD(GETNUM_1)
{
        TPNumber a(2.0, 10, 2);
        Assert::AreEqual(2.0, a.getNumber());
}

TEST_METHOD(GETNUM_2)
{
        TPNumber a(-5.0, 10, 2);
        Assert::AreEqual(-5.0, a.getNumber());
}
```

```
		TEST_METHOD(GETNUMST_1)
		{
			TPNumber a(15, 16, 2);
			Assert::AreEqual(string("F"), a.getNumberString());
		}

		TEST_METHOD(GETNUMST_2)
		{
			TPNumber a(10, 16, 2);
			Assert::AreEqual(string("A"), a.getNumberString());
		}

		TEST_METHOD(GETBASE_1)
		{
			TPNumber a(10, 16, 2);
			Assert::AreEqual(16, a.getBase());
		}

		TEST_METHOD(GETBASESTR_1)
		{
			TPNumber a(10, 16, 2);
			Assert::AreEqual(string("16"), a.getBaseString());
		}

		TEST_METHOD(GETPR_1)
		{
			TPNumber a(10, 16, 2);
			Assert::AreEqual(2, a.getPrecision());
		}

		TEST_METHOD(GETPRSTR_1)
		{
			TPNumber a(10, 16, 2);
			Assert::AreEqual(string("2"), a.getPrecisionString());
		}

		TEST_METHOD(SET_BASE_1)
		{
			TPNumber a(10, 16, 2);
			a.setBase(5);
			Assert::AreEqual(5, a.getBase());
		}

		TEST_METHOD(SET_BASE_STR_1)
		{
			TPNumber a(10, 16, 2);
			a.setBase("5");
			Assert::AreEqual(5, a.getBase());
		}

		TEST_METHOD(SET_PR_1)
		{
			TPNumber a(10, 16, 2);
			a.setPrecision(3);
			Assert::AreEqual(3, a.getPrecision());
		}

		TEST_METHOD(SET_PR_STR_1)
		{
			TPNumber a(10, 16, 2);
			a.setPrecision("3");
			Assert::AreEqual(3, a.getPrecision());
		}

	};
}
```