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
«АТД Р-ичное число»

Выполнил:
Студент группы ИП-017
Костин А.В.
Работу проверил:
ассистент кафедры ПМиК
Агалаков А.А.

Новосибирск 2023 г.

Содержание

1. Задание 3
2. Исходный код программы 11
   2.1. Код программы 11
   2.2. Код тестов 25
3. Результаты модульных тестов 33
4. Вывод 34

1. 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.

Данные:

Р-ичное число TPNumber - это действительное число (n) со знаком в системе счисления с основанием (b) (в диапазоне 2..16), содержащее целую и дробную части.

Точность представления числа – (c >= 0). Р-ичные числа неизменяемые.

Операции:

Операции могут вызываться только объектом р-ичное число (тип TPNumber), указатель на который в них передаётся по умолчанию. При описании операций этот объект называется this «само число».

КонструкторЧисло	
Вход:	Вещественное число (a), основание системы счисления (b), точность представления числа (c)
Предусловия:	Основание системы счисления (b) должно принадлежать интервалу [2..16], точность представления числа c >= 0.
Процесс:	Инициализирует поля объекта this р-ичное число:

	система счисления (b), точность представления (c). В поле (n) числа заносится (a). Например: TPNumber(a,3,3) = число a в системе счисления с тремя разрядами после троичной точки. TPNumber (a,3,2) = число a в системе счисления с двумя разрядами после троичной точки.
Постусловия:	Объект инициализирован начальными значениями.
Выход:	Нет.
КонструкторСтрока	
Вход:	Строковые представления: p–ичного числа (a), основания системы счисления (b), точности представления числа (c)
Предусловия:	Основание системы счисления (b) должно принадлежать интервалу [2..16], точность представления числа c >= 0.
Процесс:	Инициализирует поля объекта this p-ичное число: основание системы счисления (b), точностью

	представления (c). В поле (n) числа this заносится результат преобразования строки (a) в числовое представление. b-ичное число (a) и основание системы счисления (b) представлены в формате строки. Например: TPNumber (“20”,”3”,”6”) = 20 в системе счисления 3, точность 6 знаков после запятой. TPNumber (“0”,”3”,”8”) = 0 в системе счисления 3, точность 8 знаков после запятой.
Постусловия:	Объект инициализирован начальными значениями.
Выход:	Нет.
Копировать:	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт копию самого числа this (тип TPNumber)
Постусловия:	Нет.
Выход:	p-ичное число.
Сложить	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа this.
Предусловия:	Нет

Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное сложением полей (n) самого числа this и числа d.
Постусловия:	Нет
Выход:	p-ичное число.
Умножить	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа this.
Предусловия:	Нет
Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное умножением полей (n) самого числа this и числа d
Постусловия:	Нет
Выход:	P-ичное число (тип TPNumber)
Вычесть	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа this.
Предусловия:	Нет
Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное вычитанием полей (n) самого числа this и числа d
Постусловия:	Нет

Выход:	P-ичное число (тип TPNumber)
Делить	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа this.
Предусловия:	Нет
Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное делением полей (n) самого числа this и числа d
Постусловия:	Нет
Выход:	P-ичное число (тип TPNumber)
Обратить	
Вход:	Нет
Предусловия:	Поле (n) самого числа не равно 0.
Процесс:	Создаёт p-ичное число, в поле (n) которого заносится значение, полученное как 1/(n) самого числа this.
Постусловия:	Нет
Выход:	P-ичное число (тип TPNumber)
Квадрат	
Вход:	Нет
Предусловия:	Нет
Процесс:	Создаёт p-ичное число, в поле (n) которого заносится значение, полученное как квадрат поля (n) самого числа this.
Постусловия:	Нет

Выход:	P-ичное число (тип TPNumber)
ВзятьРСтрока	
Вход:	Нет
Предусловия:	Нет
Процесс:	Возвращает p-ичное число (q) в формате строки, изображающей значение поля (n) самого числа this в системе счисления (b) с точностью (c).
Постусловия:	Нет
Выход:	Строка
ВзятьРЧисло	
Вход:	Нет
Предусловия:	Нет
Процесс:	Возвращает значение поля (n) самого числа this
Постусловия:	Нет
Выход:	Вещественное значение
ВзятьОснованиеЧисло	
Вход:	Нет
Предусловия:	Нет
Процесс:	Возвращает значение поля (b) самого числа this
Постусловия:	Нет
Выход:	Целочисленное значение
ВзятьОснованиеСтрока	
Вход:	Нет
Предусловия:	Нет
Процесс:	Возвращает значение поля (b) самого числа this в формате строки, изображающей (b) в

	десятичной системе счисления.
Постусловия:	Нет
Выход:	Строка
ВзятьТочностьЧисло	
Вход:	Нет
Предусловия:	Нет
Процесс:	Возвращает значение поля (c) самого числа this
Постусловия:	Нет
Выход:	Целочисленное значение
ВзятьТочностьСтрока	
Вход:	Нет
Предусловия:	Нет
Процесс:	Возвращает значение поля (c) самого числа this в формате строки, изображающей (c) в десятичной системе счисления.
Постусловия:	Нет
Выход:	Строка
УстановитьОснованиеЧисло	**УстановитьОснованиеЧисло**
Вход:	Целое число (newb).
Предусловия:	2 <= newb <= 16
Процесс:	Устанавливает в поле (b) самого числа this значение (newb).
Постусловия:	Нет
Выход:	Нет
УстановитьОснованиеСтрока	
Вход:	Строка (bs), изображающая основание (b) р-ичного числа в десятичной системе счисления.

Предусловия:	Допустимый диапазон числа, изображаемого строкой (bs) - 2,,16
Процесс:	Устанавливает значение поля (b) самого числа this значением, полученным в результате преобразования строки (bs)
Постусловия:	Нет
Выход:	Строка.
УстановитьТочностьЧисло	
Вход:	Целое число (newc).
Предусловия:	newc >= 0.
Процесс:	Устанавливает в поле (c) самого числа значение (newc)
Постусловия:	Нет
Выход:	Нет
УстановитьТочностьСтрока	
Вход:	Строка (newc).
Предусловия:	Строка (newc) изображает десятичное целое >= 0
Процесс:	Устанавливает в поле (c) самого числа this значение, полученное преобразованием строки (newc).
Постусловия:	Нет
Выход:	Нет

2. Исходный код программы

2.1. Код программы

TPNumber.h

```
#pragma once
#include <iostream>
#include <string>

using namespace std;

class TPNumber
{
	double number;
	int base;
	int precision;
	double convertToDecimal(double number);
	double convertToDecimal(string number);

	double convertToBaseDouble(double n);
	string convertToBaseString(double n);

	bool checkNum(double number, int base, int precision);
	bool checkNum(string number, string base, string precision);
	bool checkBase(string num_str, int base);
	bool checkPrecision(string num_str, int precision);
	bool checkStr(string num_str);
	bool checkPoint(double n);
	bool checkPoint(string n);
	bool check(double number, int base, int precision);
	bool check(string number, string base, string precision);
public:
	TPNumber(double number, int base, int precision);
	TPNumber(string num_str, string base_str, string precision_str);
	TPNumber(TPNumber& n);

	TPNumber operator+(TPNumber& n);
	TPNumber operator*(TPNumber& n);
	TPNumber operator-(TPNumber& n);
	TPNumber operator/(TPNumber& n);

	TPNumber inverse();
	TPNumber square();

	double getNumber();
```

```
        string getNumberString();
        int getBase();
        string getBaseString();
        int getPrecision();
        string getPrecisionString();

        void setBase(int base);
        void setBase(string base);
        void setPrecision(int precision);
        void setPrecision(string precision);

};
```

TPNumber.cpp

```
#include "TPNumber.h"
#include <cmath>
#include <sstream>
#include <vector>
#include <iomanip>

template <typename T>
std::string toString(T val)
{
        std::ostringstream oss;
        oss.setf(std::ios_base::fixed);
        oss << val;
        return oss.str();
}

template <typename T>
std::string toString(T val, int precision)
{
        std::ostringstream oss;
        oss.setf(std::ios_base::fixed);
        oss << setprecision(precision) << val;
        return oss.str();
}

double TPNumber::convertToDecimal(double number)
{
        double num_int = (number * pow(10, precision));
        int left = (int)(num_int / pow(10, precision));
        int right = ((int)num_int % (int)pow(10, precision));
```

```
	double result = 0;

	int i = 0;
	while (left > 0)
	{
		int tmp = left % 10;
		result += tmp * pow(base, i);
		left /= 10;
		i++;
	}

	i = precision - 1;
	int j = -1;
	while (i > -1)
	{
		int tmp = right / (int)pow(10, i);
		result += tmp * pow(base, j);
		right %= (int)pow(10, i);
		i--;
		j--;
	}

	return floor(result * pow(10, precision)) / pow(10, precision);
}

vector<string> split(string str, string delim) {
	size_t pos_start = 0, pos_end, delim_len = delim.length();
	string token;
	vector<string> res;

	while ((pos_end = str.find(delim, pos_start)) != std::string::npos) {
		token = str.substr(pos_start, pos_end - pos_start);
		pos_start = pos_end + delim_len;
		res.push_back(token);
	}

	res.push_back(str.substr(pos_start));
	return res;
}

double TPNumber::convertToDecimal(string num_str)
{
	string left, right;
	int tmp;
```

```
double result = 0;

if (precision == 0)
{
	for (int i = num_str.size() - 1; i >= 0; i--)
	{
		if (num_str[i] >= 'A' && num_str[i] <= 'F')
		{
			int move = abs('A' - num_str[i]);
			tmp = 10 + move;
		}
		else
		{
			tmp = num_str[i] - '0';
		}
		result += tmp * pow(base, num_str.size() - i - 1);
	}
	return result;
}
else if (precision > 0)
{
	vector<string> substr = split(num_str, ".");
	left = substr[0];
	right = substr[1];

	for (int i = left.size() - 1; i >= 0; i--)
	{
		if (left[i] >= 'A' && left[i] <= 'F')
		{
			int move = abs('A' - left[i]);
			tmp = 10 + move;
		}
		else
		{
			tmp = left[i] - '0';
		}
		result += tmp * pow(base, left.size() - i - 1);
	}

	for (int i = 0; i < right.size(); i++)
	{
		if (right[i] >= 'A' && right[i] <= 'F')
		{
			int move = abs('A' - right[i]);
```

```
                        tmp = 10 + move;
                }
                else
                {
                        tmp = right[i] - '0';
                }
                result += tmp * pow(base, -(i + 1));
        }


        return floor(result * pow(10, precision)) / pow(10, precision);
    }
    else
    {
        return -1;
    }
}

string reverse(string str) {
    string res = "";
    for (int i = str.size() - 1; i >= 0; --i)
        res += str[i];
    return res;
}

double TPNumber::convertToBaseDouble(double num)
{
    try {
        if (base > 1 && base < 10 && num != 0) {
            string num_str = toString(num, precision);
            int j;
            for (j = 0; j < num_str.length() && num_str[j] != '.'; j++);

            if (j < num_str.length()) {
                vector<string> num_str_split = split(num_str, ".");
                int left = stoi(num_str_split[0]);
                double right;
                if (num_str_split[1].length() < precision) {
                    right = stod(num_str_split[1].substr(0, precision -
1));
                }
                else {
                    right = stod(num_str_split[1].substr(0, precision));
                }
```

```
                string result = "";

                while (left > 0)
                {
                        int tmp = left % base;
                        result += toString(tmp);
                        left /= base;
                }
                result = reverse(result);

                result += ".";
                string sub_res = "";
                string right_str = "0." + toString(right);
                int i = 0;
                while (i < precision + 1) {
                        right = stod(right_str);
                        right *= base;
                        right_str = toString(right);
                        for (j = 0; j < right_str.length() && right_str[j] !=
'.'; j++);
                        if (j < right_str.length()) {
                                vector<string> sp = split(right_str, ".");
                                sub_res += sp[0];
                                right_str = "0." + right_str.substr(2);
                        }
                        else {
                                sub_res += right_str;
                                right_str = "0.0";
                        }
                        i++;
                }
                result += sub_res;
                double res_double = stod(result);
                return floor(res_double * pow(10, precision)) / pow(10,
precision);
        }
        else {
                int left = stoi(num_str);
                string result = "";
                while (left > 0)
                {
                        int tmp = left % base;
                        result += toString(tmp);
                        left = left / base;
```

```
                }
                result = reverse(result);

                return stod(result);
            }
        }
        else if (num == 0.0) {
            return 0.0;
        }
        else {
            return -1;
        }
    }
    catch (const std::exception& e)
    {
        throw e.what();
    }
}

string TPNumber::convertToBaseString(double num)
{
    try {
        if (base > 1 && base < 10) {
            string result = toString(convertToBaseDouble(num));
            return result;
        }
        else if (base > 10 && base < 17) {
            if (abs(num - 0.0) < 0.001) {
                return "0.0";
            }

            string num_str = toString(num);
            if (checkPoint(num_str)) {
                vector<string> spliter = split(num_str, ".");
                int left = stoi(spliter[0]);
                double right = stod(spliter[1]);
                string result = "";

                while (left > 0) {
                    double tmp = left % base;
                    char tmp_char = toString(tmp)[0];
                    if (tmp > 9) {
                        tmp_char = 'A' + tmp - 10;
                    }
```

```
                result += tmp_char;
                left /= base;
            }
            result = reverse(result) + ".";

            int iter = 0;
            double tmp_right = right, iter_right = 0;
            while ((int)floor(tmp_right) > 0) {
                tmp_right /= 10;
                iter_right++;
            }
            right /= pow(10, iter_right);
            while (iter < precision) {
                right *= base;
                int add = (int)floor(right);
                char add_char = toString(add)[0];
                if (add > 9) {
                    add_char = 'A' + add - 10;
                }
                result += add_char;
                right = right - floor(right);
                iter++;
            }
            return result;
        }
        else {
            int left = stoi(num_str);
            string result = "";
            while (left > 0) {
                double tmp = left % base;
                char tmp_char = toString(tmp)[0];
                if (tmp > 9) {
                    tmp_char = 'A' + tmp - 10;
                }
                result += tmp_char;
                left /= base;
            }
            result = reverse(result);
            return result;
        }
    }
    else if (base == 10) {
        return toString(number, 1);
    }
```

```
	}
	catch (const std::exception& e)
	{
		throw e.what();
	}
	return nullptr;
}

bool TPNumber::checkNum(double number, int base, int precision)
{
	string num_str = toString(number);
	return checkBase(num_str, base) && checkPrecision(num_str, precision);
}

bool TPNumber::checkNum(string num_str, string base_str, string precision_str)
{
	int base = stoi(base_str);
	int precision = stoi(precision_str);
	return checkBase(num_str, base) && checkPrecision(num_str, precision)
&& checkStr(num_str);
}

bool TPNumber::checkBase(string num_str, int base)
{
	for (char c : num_str)
	{
		if (c == '.')
			continue;
		int move = abs('A' - c);
		int digit = c - '0';
		if (c >= 'A' && c <= 'Z') {
			digit = 10 + move;
		}
		if (digit >= base)
			return false;
	}
	return true;
}

bool TPNumber::checkPrecision(string num_str, int precision)
{
	int real_precision = 0;
	bool point_met = false;
	for (char c : num_str) {
```

```
            if (c == '.' && !point_met) {
                point_met = true;
                continue;
            }
            if (point_met)
                real_precision++;
            if (real_precision > precision && c != '0')
                return false;
        }
        return true;
}

bool TPNumber::checkStr(string num_str)
{
        bool point_met = false;
        for (char c : num_str) {
            if (c == '.' && !point_met) {
                point_met = true;
                continue;
            }
            if (!((c >= '0' && c <= '9') || (c >= 'A' && c <= 'F'))) {
                return false;
            }
        }
        return true;
}

bool TPNumber::checkPoint(double n)
{
        string n_str = toString(n);
        return checkPoint(n_str);
}

bool TPNumber::checkPoint(string n_str)
{
        int i;
        for (i = 0; i < n_str.length() && n_str[i] != '.'; i++)
            ;
        if (i < n_str.length())
            return true;
        return false;
}

bool TPNumber::check(double number, int base, int precision)
```

```
{
	string num_str = toString(number);
	if (!checkStr(num_str))
		return false;
	if (!checkBase(num_str, base))
		return false;
	if (!checkPrecision(num_str, precision))
		return false;
	return true;
}

bool TPNumber::check(string num_str, string base_str, string prec_str)
{
	int base = stoi(base_str);
	int precision = stoi(prec_str);
	if (!checkStr(num_str))
		return false;
	if (!checkBase(num_str, base))
		return false;
	if (!checkPrecision(num_str, precision))
		return false;
	return true;
}

TPNumber::TPNumber(double number, int base, int precision)
{
	try {
		if (base < 10 && base > 1 && precision >= 0 && check(number,
base, precision)) {
			this->base = base;
			this->precision = precision;
			this->number = convertToDecimal(number);
		}
		else if (base == 10) {
			this->number = number;
			this->base = base;
			this->precision = precision;
		}
		else {
			this->number = 0;
			this->base = 10;
			this->precision = 0;
		}
	}
```

```
        catch (const std::exception& e) {
                throw e.what();
        }
}

TPNumber::TPNumber(string num_str, string base_str, string precision_str)
{
        try
        {
                this->base = stoi(base_str);
                this->precision = stoi(precision_str);

                if (this->base <= 16 && this->base > 1 && this->base != 10 && this-
>precision >= 0 && check(num_str, base_str, precision_str)) {
                        number = this->convertToDecimal(num_str);
                }
                else if (base == 10) {
                        this->number = stod(num_str);
                }
                else {
                        this->number = 0;
                        this->base = 10;
                        this->precision = 0;
                }
        }
        catch (const std::exception& e)
        {
                throw e.what();
        }
}

TPNumber::TPNumber(TPNumber& n)
{
        number = n.number;
        base = n.base;
        precision = n.precision;
}

TPNumber TPNumber::operator+(TPNumber& n)
{
        TPNumber tmp = n;
        if (n.base != base || n.precision != precision) {
                tmp.number = 0;
                return tmp;
```

```
	}
	tmp.number = number + n.number;
	return tmp;
}

TPNumber TPNumber::operator*(TPNumber& n)
{
	TPNumber tmp = n;
	if (n.base != base || n.precision != precision) {
		tmp.number = 0;
		return tmp;
	}
	tmp.number = number * n.number;
	return tmp;
}

TPNumber TPNumber::operator-(TPNumber& n)
{
	TPNumber tmp = n;
	if (n.base != base || n.precision != precision) {
		tmp.number = 0;
		return tmp;
	}
	tmp.number = number - n.number;
	return tmp;
}

TPNumber TPNumber::operator/(TPNumber& n)
{
	TPNumber tmp = n;
	if (n.base != base || n.precision != precision || n.number == 0) {
		tmp.number = 0;
		return tmp;
	}
	tmp.number = number / n.number;
	return tmp;
}

TPNumber TPNumber::inverse()
{
	TPNumber tmp = *this;
	tmp.number = 1.0 / number;
	return tmp;
}
```

```
TPNumber TPNumber::square()
{
	TPNumber tmp = *this;
	tmp.number = number * number;
	return tmp;
}

double TPNumber::getNumber()
{
	return convertToBaseDouble(number);
}

string TPNumber::getNumberString()
{
	return convertToBaseString(number);
}

int TPNumber::getBase()
{
	return base;
}

string TPNumber::getBaseString()
{
	return toString(base);
}

int TPNumber::getPrecision()
{
	return precision;
}

string TPNumber::getPrecisionString()
{
	return toString(precision);
}

void TPNumber::setBase(int base)
{
	if (check(number, base, precision)) {
		this->base = base;
	}
}
```

```
void TPNumber::setBase(string base_str)
{
	try {
		int base = stoi(base_str);
		if (check(number, base, precision)) {
			this->base = base;
		}
	}
	catch (const std::exception& e) {
		throw e.what();
	}
}

void TPNumber::setPrecision(int precision)
{
	if (check(number, base, precision)) {
		this->precision = precision;
	}
}

void TPNumber::setPrecision(string precision_str)
{
	try {
		int precision = stoi(precision_str);
		if (check(number, base, precision)) {
			this->precision = precision;
		}
	}
	catch (const std::exception& e) {
		throw e.what();
	}
}
```

2.2. Код тестов

UnitTest.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../ModernCodingPNumber/TPNumber.h"
#include <iostream>
#include <string>
using namespace std;

using namespace Microsoft::VisualStudio::CppUnitTestFramework;
```

```
namespace UnitTest
{
	TEST_CLASS(UnitTest)
	{
	public:
		TEST_METHOD(TestCtorSuccess)
		{
			double a = 1011.1011;
			int b = 2;
			int c = 4;

			double extend = 1011.1011;
			TPNumber pnum(a, b, c);
			pnum.getNumber();
			double result = pnum.getNumber();
			Assert::AreEqual(extend, result);
		}
		TEST_METHOD(Test)
		{
			double a = 1011.1011;
			int b = 2;
			int c = 4;

			double extend = 1011.1011;
			TPNumber pnum(a, b, c);
			pnum.getNumber();
			double result = pnum.getNumber();
			Assert::AreEqual(extend, result);
		}
		TEST_METHOD(TestCtorFailPrecision)
		{
			double a = 1011.1010;
			int b = 2;
			int c = -1;

			double extend = 0.0;
			TPNumber pnum(a, b, c);
			double result = pnum.getNumber();
			Assert::AreEqual(extend, result);
		}
		TEST_METHOD(TestCtorFailBase)
		{
			double a = 1011.1010;
```

```
        int b = 1;
        int c = 4;

        double extend = 0.0;
        TPNumber pnum(a, b, c);
        double result = pnum.getNumber();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestCtorString)
{
        string a = "ABC123.435DC";
        string b = "16";
        string c = "5";

        string extend = "ABC123.435D2";
        TPNumber pnum(a, b, c);
        string result = pnum.getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestCtorStrFailPrecision)
{
        string a = "ABC123.435DC";
        string b = "16";
        string c = "3";

        string extend = "0.0";
        TPNumber pnum(a, b, c);
        string result = pnum.getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestCtorStrFailBase)
{
        string a = "ABC123.435DC";
        string b = "12";
        string c = "5";

        string extend = "0.0";
        TPNumber pnum(a, b, c);
        string result = pnum.getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestCtorStrFailNum)
{
        string a = "ABJ123.435DC";
```

```
        string b = "16";
        string c = "5";

        string extend = "0.0";
        TPNumber pnum(a, b, c);
        string result = pnum.getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestAdd2)
{
        string a = "1110101.110101";
        string b = "2";
        string c = "6";
        string a1 = "111101.100001";
        string b1 = "2";
        string c1 = "6";

        string extend = "10110011.010110";
        TPNumber pnum(a, b, c);
        TPNumber pnum1(a1, b1, c1);
        string result = (pnum + pnum1).getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestAdd15)
{
        string a = "1837A.342B";
        string b = "15";
        string c = "4";
        string a1 = "34C01.DDA1";
        string b1 = "15";
        string c1 = "4";

        string extend = "4D07C.22C6";
        TPNumber pnum(a, b, c);
        TPNumber pnum1(a1, b1, c1);
        string result = (pnum + pnum1).getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestAddDiffBase)
{
        string a = "1837A.342B";
        string b = "16";
        string c = "4";
        string a1 = "34C01.DDA1";
```

```
        string b1 = "15";
        string c1 = "4";

        string extend = "0.0";
        TPNumber pnum(a, b, c);
        TPNumber pnum1(a1, b1, c1);
        string result = (pnum + pnum1).getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestAddDiffPrec)
{
        string a = "1837A.342B";
        string b = "16";
        string c = "4";
        string a1 = "34C01.DDA1A";
        string b1 = "16";
        string c1 = "5";

        string extend = "0.0";
        TPNumber pnum(a, b, c);
        TPNumber pnum1(a1, b1, c1);
        string result = (pnum + pnum1).getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestMult)
{
        string a = "1283.22";
        string b = "15";
        string c = "2";
        string a1 = "34.34";
        string b1 = "15";
        string c1 = "2";

        string extend = "3C877.E8";
        TPNumber pnum(a, b, c);
        TPNumber pnum1(a1, b1, c1);
        string result = (pnum * pnum1).getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestSub)
{
        string a = "1283.22";
        string b = "15";
        string c = "2";
```

```
        string a1 = "34.34";
        string b1 = "15";
        string c1 = "2";

        string extend = "124D.DE";
        TPNumber pnum(a, b, c);
        TPNumber pnum1(a1, b1, c1);
        string result = (pnum - pnum1).getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestDiv)
{
        string a = "1283.22";
        string b = "15";
        string c = "2";
        string a1 = "34.34";
        string b1 = "15";
        string c1 = "2";

        string extend = "55.36";
        TPNumber pnum(a, b, c);
        TPNumber pnum1(a1, b1, c1);
        string result = (pnum / pnum1).getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestDivZero)
{
        string a = "1283.22";
        string b = "15";
        string c = "2";
        string a1 = "0.0";
        string b1 = "15";
        string c1 = "2";

        string extend = "0.0";
        TPNumber pnum(a, b, c);
        TPNumber pnum1(a1, b1, c1);
        string result = (pnum / pnum1).getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestInverse)
{
        string a = "1283.22";
        string b = "15";
```

```
        string c = "2";

        string extend = "0.0";
        TPNumber pnum(a, b, c);
        string result = pnum.inverse().getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestSqr)
{
        string a = "1283.22";
        string b = "15";
        string c = "2";

        string extend = "157D924.6D";
        TPNumber pnum(a, b, c);
        string result = pnum.square().getNumberString();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestSetBaseFail)
{
        string a = "1283.22";
        string b = "15";
        string c = "2";

        int extend = 15;
        TPNumber pnum(a, b, c);
        pnum.setBase(2);
        int result = pnum.getBase();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestSetBase)
{
        string a = "1283.22";
        string b = "15";
        string c = "2";

        int extend = 16;
        TPNumber pnum(a, b, c);
        pnum.setBase(16);
        int result = pnum.getBase();
        Assert::AreEqual(extend, result);
}
TEST_METHOD(TestSetPrec)
{
```

```
            string a = "1283.22";
            string b = "15";
            string c = "2";

            int extend = 4;
            TPNumber pnum(a, b, c);
            pnum.setPrecision(4);
            int result = pnum.getPrecision();
            Assert::AreEqual(extend, result);
        }
        TEST_METHOD(TestSetFail)
        {
            string a = "1283.1337";
            string b = "15";
            string c = "4";

            int extend = 4;
            TPNumber pnum(a, b, c);
            pnum.setPrecision(2);
            int result = pnum.getPrecision();
            Assert::AreEqual(extend, result);
        }
    };
}
```

3. Результаты модульных тестов

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.