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
«Абстрактный тип р-ичное число»

Выполнил:
Студент группы ИП-911
Мироненко К.А.
Работу проверил:
доцент кафедры ПМиК
Зайцев М.Г.

Новосибирск 2022 г.

СОДЕРЖАНИЕ

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.

Спецификация типа данных «р-ичное число».

ADT TPNumber

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], точностьпредставления числа c >= 0.

Процесс:	Инициализирует поля объекта this p-ичное число: система счисления (b), точность представления (c). В поле (n) числа заносится (a). Например: TPNumber(a,3,3) = число a в системе счисления 3 стремя разрядами после троичной точки. TPNumber (a,3,2) = число a в системе счисления 3c 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], точностьпредставления 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 системесчисления 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).
Выход:	p-ичное число.
Постусловия:	Нет.
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.
Выход:	p-ичное число.
Постусловия:	Нет
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.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
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 (типTPNumber), полученное вычитанием полей (n) самого числа this и числа d.

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

Процесс:	Создаёт p-ичное число, в поле (n) которогозаносится значение, полученное как квадрат поля (n) самого числа this.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
ВзятьРЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (n) самого числа this.
Выход:	Вещественное значение.
Постусловия:	Нет.
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).
Выход:	Строка.
Постусловия:	Нет.
ВзятьОснованиеЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (b) самого числа this
Выход:	Целочисленное значение

Постусловия:	Нет.
ВзятьОснованиеСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (b) самого числа this вформате строки, изображающей (b) в десятичной системе счисления.
Выход:	Строка.
Постусловия:	Нет.
ВзятьТочностьЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (c) самого числа this.
Выход:	Целое значение.
Постусловия:	Нет.
ВзятьТочностьСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (c) самого числа this в формате строки, изображающей (c) в десятичнойсистеме счисления.
Выход:	Строка.
Постусловия:	Нет.
УстановитьОснованиеЧисло	

Вход:	Целое число (newb).
Предусловия:	2 <= newb <= 16.
Процесс:	Устанавливает в поле (b) самого числа this значение (newb).
Выход:	Нет.
Постусловия:	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 thisзначением, полученным в результате преобразования строки (bs).
Выход:	Строка.
Постусловия:	Нет.
УстановитьТочностьЧисло	
Вход:	Целое число (newc).
Предусловия:	newc >= 0.
Процесс:	Устанавливает в поле (c) самого числа значение (newc).
Выход:	Нет.
Постусловия:	Нет.

УстановитьТочностьСтрока	
Вход:	Строка (newc).
Предусловия:	Строка (newc) изображает десятичное целое >= 0.
Процесс:	Устанавливает в поле (c) самого числа thisзначение, полученное преобразованием строки (newc).
Выход:	Нет.
Постусловия:	Нет.

end TPNumber

2. Исходный код программы
2.1. Код программы

Program.cs

```
using System;

namespace numeral

{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

TPNumber.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace numeral
{
    // Обработка исключения
    public class MyException : Exception
    {
        public MyException(string str) : base(str) { }
    }
    public abstract class TPNumber
    {
        protected double n;   // Number
        protected int b;      // Base
        protected int c;      // Correctness
        public TPNumber()
        {
            this.n = 0f;
            this.b = 10;
            this.c = 0;
        }
        public TPNumber(double a, int b, int c)
        {
            try
            {
                if (b < 10 && b > 1 && c >= 0 && check(a, b, c))
                {
                    this.b = b;
                    this.c = c;
                    n = ConvertToDouble(a);
                }
                else if (b == 10)
                {
                    this.b = b;
                    this.c = c;
                    n = a;
                }
                else
                {
                    this.n = 0f;
                    this.b = 10;
```

```
                this.c = 0;
            }
        }
        catch
        {
            throw new MyException("Error");

        }

    }
    public TPNumber(string a, string b, string c)
    {
        this.b = Convert.ToInt32(b);
        this.c = Convert.ToInt32(c);
        try
        {
            if (this.b < 17 && this.b > 1 && this.b != 10 && this.c >= 0 && check(a,
b, c))
            {
                n = ConvertStringToDouble(a);
            }
            else if (this.b == 10)
            {
                n = Convert.ToDouble(a);
            }
        }
        catch
        {
            throw new MyException("Error");
        }

    }
    public TPNumber(TPNumber d)
    {
        n = d.n;
        b = d.b;
        c = d.c;
    }
    public TPNumber Copy()
    {
        return (TPNumber)this.MemberwiseClone();
    }
    public TPNumber Add(TPNumber d)
    {
        TPNumber tmp = d.Copy();
        if (d.b != this.b || d.c != this.c)
        {
            tmp.n = 0.0;
            return tmp;
        }
        tmp.n = this.n + d.n;
        return tmp;
    }
    public TPNumber Mult(TPNumber d)
    {
        TPNumber tmp = d.Copy();
        if (d.b != this.b || d.c != this.c)
        {
            tmp.n = 0.0;
            return tmp;
        }
        tmp.n = this.n * d.n;
        return tmp;
    }
```

```
public TPNumber Substract(TPNumber d)
{
    TPNumber tmp = d.Copy();
    if (d.b != this.b || d.c != this.c)
    {
        tmp.n = 0.0;
        return tmp;
    }
    tmp.n = this.n - d.n;
    return tmp;
}
public TPNumber Del(TPNumber d)
{
    TPNumber tmp = d.Copy();
    if (d.b != this.b || d.c != this.c)
    {
        tmp.n = 0.0;
        return tmp;
    }
    tmp.n = this.n / d.n;
    return tmp;
}
public TPNumber Revers()
{
    TPNumber tmp = this.Copy();
    tmp.n = 1 / this.n;
    return tmp;
}
public TPNumber Sqr()
{
    TPNumber tmp = this.Copy();
    tmp.n = this.n * this.n;
    return tmp;
}
public double GetPNumber()
{
    return ConvertDoubleToBaseDouble(n);
}
public string GetPString()
{
    return ConvertStringToBaseDouble(n);
}
public int GetBaseNumber()
{
    return this.b;
}
public string GetBaseString()
{
    return this.b.ToString();
}
public int GetCorrectnessNumber()
{
    return this.c;
}
public string GetCorrectnessString()
{
    return this.c.ToString();
}
public void SetBaseNumber(int b)
{
    if (check(this.n, b, this.c))
    {
        this.b = b;
    }
```

```
    else
    {
        return;
    }
}
public void SetBaseString(string b)
{
    if (check(this.n, Convert.ToInt32(b), this.c))
    {
        this.b = Convert.ToInt32(b);
    }
    else
    {
        return;
    }

}
public void SetCorrectnessNumber(int c)
{
    if (check(this.n, this.b, c))
    {
        this.c = c;
    }
    else
    {
        return;
    }

}
public void SetCorrectnessString(string c)
{
    if (check(this.n, this.b, Convert.ToInt32(c)))
    {
        this.c = Convert.ToInt32(c);
    }
    else
    {
        return;
    }

}
private double ConvertToDouble(double a)
{
    double num_int = (a * Math.Pow(10, c));
    int left = (int)(num_int / Math.Pow(10, c));
    int right = (int)(num_int % (int)Math.Pow(10, c));
    double result = 0;

    int i = 0;
    while (left > 0)
    {
        int tmp = left % 10;
        result += tmp * Math.Pow(b, i);
        left /= 10;
        i++;
    }

    i = c - 1;
    int j = -1;
    while (i > -1)
    {
        int tmp = right / (int)Math.Pow(10, i);
        result += tmp * Math.Pow(b, j);
        right %= (int)Math.Pow(10, i);
```

```
        i--;
        j--;
    }

    return Math.Floor(result * Math.Pow(10, c)) / Math.Pow(10, c); ;
}
private double ConvertStringToDouble(string str)
{
    string left, right;
    int tmp;
    double result = 0;

    if (c == 0)
    {
        for (int i = str.Length - 1; i >= 0; i--)
        {
            if (str[i] >= 'A' && str[i] <= 'Z')
            {
                int move = Math.Abs('A' - str[i]);
                tmp = 10 + move;
            }
            else
            {
                tmp = str[i] - '0';
            }
            result += tmp * Math.Pow(b, str.Length - i - 1);
        }
        return result;
    }
    else if (c > 0)
    {
        string[] substr = str.Split(",");
        left = substr[0];
        right = substr[1];

        for (int i = left.Length - 1; i >= 0; i--)
        {
            if (left[i] >= 'A' && left[i] <= 'Z')
            {
                int move = Math.Abs('A' - left[i]);
                tmp = 10 + move;
            }
            else
            {
                tmp = left[i] - '0';
            }
            result += tmp * Math.Pow(b, left.Length - i - 1);
        }

        for (int i = 0; i < right.Length; i++)
        {
            if (right[i] >= 'A' && right[i] <= 'Z')
            {
                int move = Math.Abs('A' - right[i]);
                tmp = 10 + move;
            }
            else
            {
                tmp = right[i] - '0';
            }
            result += tmp * Math.Pow(b, -(i + 1));
        }
```

```
                return Math.Floor(result * Math.Pow(10, c)) / Math.Pow(10, c);
            }
            else
            {
                return -1;
            }
        }
        private double ConvertDoubleToBaseDouble(double a)
        {
            if (b > 1 && b < 10 && a != 0)
            {
                string num_10_str = a.ToString();
                int j;
                for (j = 0; j < num_10_str.Length && num_10_str[j] != ','; j++) { }

                if (j < num_10_str.Length)
                {
                    string[] num_10_str_split = num_10_str.Split(",");
                    int left = Convert.ToInt32(num_10_str_split[0]);
                    double right;
                    if (num_10_str_split[1].Length < c)
                    {
                        right = Convert.ToDouble(num_10_str_split[1].Substring(0, this.c
- 1));
                    }
                    else
                    {
                        right = Convert.ToDouble(num_10_str_split[1].Substring(0,
this.c));
                    }
                    string result = "";

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

                    result = Revers(result);

                    result += ",";
                    string sub_res = "";
                    string right_str = "0," + right;
                    int i = 0;
                    while (i < c + 1)
                    {
                        right = Convert.ToDouble(right_str);
                        right *= (double)b;
                        right_str = right.ToString();
                        for (j = 0; j < right_str.Length && right_str[j] != ','; j++) { }
                        if (j < right_str.Length)
                        {
                            string[] sp = right_str.Split(",");
                            sub_res += sp[0];
                            right_str = "0," + right_str.Substring(2);
                        }
                        else
                        {
                            sub_res += right_str;
                            right_str = "0,0";
                        }

                        i++;
```

```
            }
            result += sub_res;
            double res_double = Convert.ToDouble(result);
            res_double = Math.Round(res_double, c, MidpointRounding.ToZero);

            return res_double;
        }
        else
        {
            int left = Convert.ToInt32(num_10_str);

            string result = "";

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

            result = Revers(result);

            return Convert.ToDouble(result);
        }
    }
    else if (a == 0)
    {
        return 0.0;
    }
    else
    {
        return -1;
    }
}
private string ConvertStringToBaseDouble(double n)
{
    try
    {
        if (b > 1 && b < 10)
        {
            string result = ConvertDoubleToBaseDouble(n).ToString();
            return result;
        }
        else if (b > 10 && b < 17)
        {
            if (Math.Abs(n - 0.0) < 0.001)
            {
                return "0,0";
            }
            string number = n.ToString();
            if (checkPoint(number))
            {
                string[] spliter = number.Split(',');
                int left = Convert.ToInt32(spliter[0]);
                double right = Convert.ToDouble(spliter[1]);
                string result = "";

                while (left > 0)
                {
                    double tmp = left % this.b;
                    char tmp_char = tmp.ToString().ToCharArray()[0];
                    if (tmp > 9)
                    {
                        tmp_char = (char)('A' + tmp - 10);
```

```
                    }
                    result += tmp_char;
                    left /= b;
                }
                result = Revers(result) + ",";

                int iter = 0;
                double tmp_right = right, iter_right = 0;
                for (; Math.Truncate(tmp_right) > 0; iter_right++)
                {
                    tmp_right /= 10;
                }
                right = right / Math.Pow(10, iter_right);
                while (iter < c)
                {
                    right *= b;
                    int add = (int)Math.Truncate(right);
                    char add_char = add.ToString().ToCharArray()[0];
                    if (add > 9)
                    {
                        add_char = (char)('A' + add - 10);
                    }
                    result += add_char;
                    right = right - Math.Truncate(right);
                    iter++;
                }

                return result;
            }
            else
            {
                int left = Convert.ToInt32(number);
                string result = "";
                while (left > 0)
                {
                    double tmp = left % this.b;
                    char tmp_char = tmp.ToString().ToCharArray()[0];
                    if (tmp > 9)
                    {
                        tmp_char = (char)('A' + tmp - 10);
                    }
                    result += tmp_char;
                    left /= b;
                }
                result = Revers(result);
                return result;
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
    return null;
}
private bool checkPoint(string n)
{
    int i;
    for (i = 0; i < n.Length && n[i] != ','; i++) { }
    if (i < n.Length)
    {
        return true;
    }
    return false;
```

```
}
private bool checkPoint(double n)
{
    string n_str = n.ToString();
    int i;
    for (i = 0; i < n_str.Length && n_str[i] != ','; i++) { }
    if (i < n_str.Length)
    {
        return true;
    }
    return false;
}
private string Revers(string str)
{
    char[] sub_char = str.ToCharArray();
    for (int j = 0; j < str.Length / 2; j++)
    {
        char tmp = sub_char[j];
        sub_char[j] = sub_char[sub_char.Length - j - 1];
        sub_char[sub_char.Length - j - 1] = tmp;
    }

    string result = "";
    for (int j = 0; j < sub_char.Length; j++)
    {
        result += sub_char[j];
    }
    return result;
}
private bool checkOnBase(string a, int b)
{
    foreach (char iter in a)
    {
        int move = Math.Abs('A' - iter);
        int iter_int = iter - '0';
        if (iter >= 'A' && iter <= 'Z')
        {
            iter_int = 10 + move;
        }
        if (iter == ',')
        {
            continue;
        }
        if (iter_int >= b)
        {
            return false;
        }
    }
    return true;
}
private bool checkOnC(string a, int c)
{
    if (checkPoint(a) && c > 0)
    {
        string[] spliter = a.Split(',');
        if (spliter[1].Length == c)
        {
            return true;
        }
    }
    return false;
}
private bool checkOnSymbol(string a)
{
```

```
            foreach (char iter in a)
            {
                if (iter >= 'a' && iter <= 'z')
                {
                    return false;
                }
            }
            return true;
        }
        private bool check(double a, int b, int c)
        {
            string a_str = a.ToString();
            if (!checkOnBase(a_str, b))
            {
                return false;
            }
            if (!checkOnC(a_str, c))
            {
                return false;
            }
            if (!checkOnSymbol(a_str))
            {
                return false;
            }
            return true;
        }
        private bool check(string a, string b, string c)
        {
            int b_int = Convert.ToInt32(b);
            int c_int = Convert.ToInt32(c);
            if (!checkOnBase(a, b_int))
            {
                return false;
            }
            if (!checkOnC(a, c_int))
            {
                return false;
            }
            if (!checkOnSymbol(a))
            {
                return false;
            }
            return true;
        }
    }
}
```

PNumber.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace numeral
{
    public class PNumber : TPNumber
    {
        public PNumber() : base()
        {

        }

        public PNumber(double a, int b, int c) : base(a, b, c)
```

```
        {

        }
        public PNumber(string a, string b, string c) : base(a, b, c)
        {

        }
        public PNumber(TPNumber d) : base(d)
        {

        }
    }
}
```

2.2. Код тестов

UnitTestPNumber.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using numeral;

namespace UnitTests
{
    [TestClass]
    public class UnitPNumber
    {
        [TestMethod]
        public void TestConstructorGood()
        {
            double a = 1011.1011;
            int b = 2;
            int c = 4;

            double extend = 1011.1011;
            PNumber iP = new PNumber(a, b, c);
            double result = iP.GetPNumber();
            Assert.AreEqual(extend, result);
        }
        [TestMethod]
        public void TestConstructorFail()
        {
            double a = 1011.1010;
            int b = 2;
            int c = -1;

            double extend = 0.0;
            PNumber iP = new PNumber(a, b, c);
            double result = iP.GetPNumber();
            Assert.AreEqual(extend, result);
        }
        [TestMethod]
        public void TestConstructorFailC()
        {
            double a = 1011.1010;
            int b = 2;
            int c = -1;

            double extend = 0.0;
            PNumber iP = new PNumber(a, b, c);
            double result = iP.GetPNumber();
            Assert.AreEqual(extend, result);
        }
        [TestMethod]
        public void TestConstructorFailB()
        {
            double a = 1011.1010;
            int b = 1;
            int c = 4;

            double extend = 0.0;
            PNumber iP = new PNumber(a, b, c);
            double result = iP.GetPNumber();
            Assert.AreEqual(extend, result);
        }
        [TestMethod]
        public void TestConstructorString()
        {
            string a = "ABC123,435DC";
```

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

    string extend = "ABC123,435D2";
    PNumber iP = new PNumber(a, b, c);
    string result = iP.GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestConstructorStringFailC()
{
    string a = "ABC123,435DC";
    string b = "16";
    string c = "6";

    string extend = "0,0";
    PNumber iP = new PNumber(a, b, c);
    string result = iP.GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestConstructorStringFailB()
{
    string a = "ABC123,435DC";
    string b = "12";
    string c = "5";

    string extend = "0,0";
    PNumber iP = new PNumber(a, b, c);
    string result = iP.GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestConstructorStringFail()
{
    string a = "abc123,435ac";
    string b = "12";
    string c = "5";

    string extend = "0,0";
    PNumber iP = new PNumber(a, b, c);
    string result = iP.GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestAdd2()
{
    string a = "1110101,110101";
    string b = "2";
    string c = "6";
    string a1 = "111101,100001";
    string b1 = "2";
    string c1 = "6";

    string extend = "10110011,01011";
    PNumber iP = new PNumber(a, b, c);
    PNumber iP1 = new PNumber(a1, b1, c1);
    string result = iP.Add(iP1).GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestAdd15()
{
    string a = "1837A,342B";
```

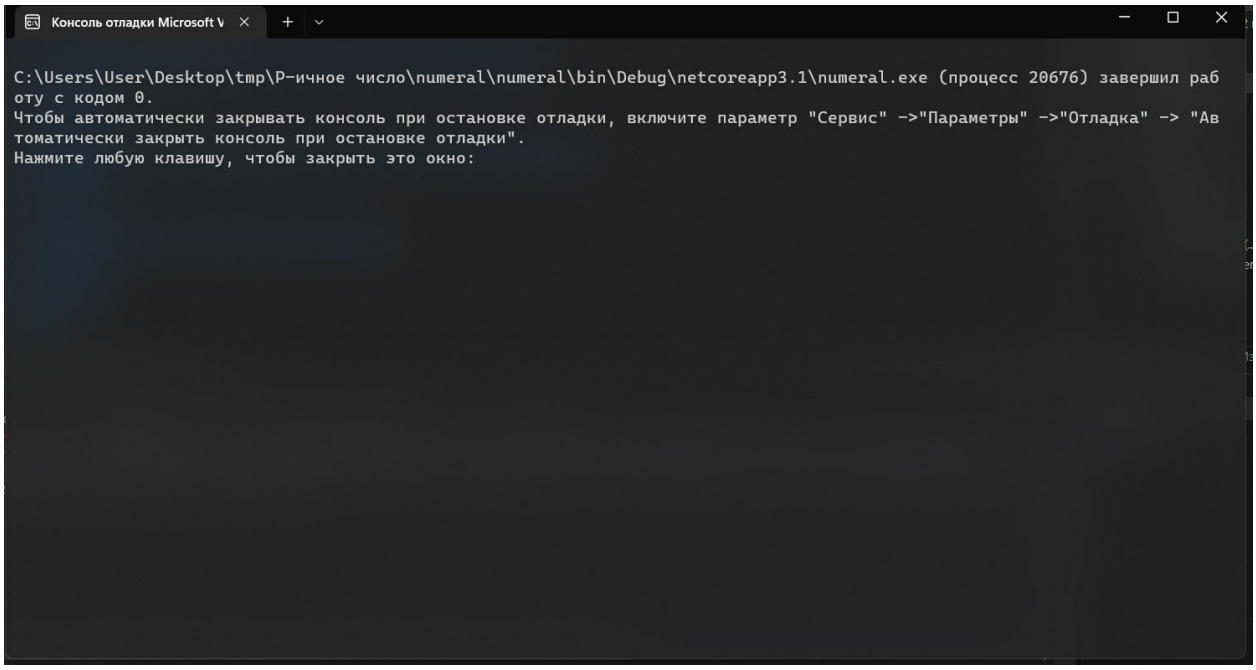
```
    string b = "15";
    string c = "4";
    string a1 = "34C01,DDA1";
    string b1 = "15";
    string c1 = "4";

    string extend = "4D07C,22C6";
    PNumber iP = new PNumber(a, b, c);
    PNumber iP1 = new PNumber(a1, b1, c1);
    string result = iP.Add(iP1).GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void 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";
    PNumber iP = new PNumber(a, b, c);
    PNumber iP1 = new PNumber(a1, b1, c1);
    string result = iP.Add(iP1).GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestAddDiffC()
{
    string a = "1837A,342B";
    string b = "16";
    string c = "4";
    string a1 = "34C01,DDA1A";
    string b1 = "15";
    string c1 = "5";

    string extend = "0,0";
    PNumber iP = new PNumber(a, b, c);
    PNumber iP1 = new PNumber(a1, b1, c1);
    string result = iP.Add(iP1).GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void 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";
    PNumber iP = new PNumber(a, b, c);
    PNumber iP1 = new PNumber(a1, b1, c1);
    string result = iP.Mult(iP1).GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestSubstract()
{
    string a = "1283,22";
```

```
    string b = "15";
    string c = "2";
    string a1 = "34,34";
    string b1 = "15";
    string c1 = "2";

    string extend = "124D,DE";
    PNumber iP = new PNumber(a, b, c);
    PNumber iP1 = new PNumber(a1, b1, c1);
    string result = iP.Substract(iP1).GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestDel()
{
    string a = "1283,22";
    string b = "15";
    string c = "2";
    string a1 = "34,34";
    string b1 = "15";
    string c1 = "2";

    string extend = "55,36";
    PNumber iP = new PNumber(a, b, c);
    PNumber iP1 = new PNumber(a1, b1, c1);
    string result = iP.Del(iP1).GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestRevers()
{
    string a = "1283,22";
    string b = "15";
    string c = "2";

    string extend = "0,0";
    PNumber iP = new PNumber(a, b, c);

    string result = iP.Revers().GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestSqrt()
{
    string a = "1283,22";
    string b = "15";
    string c = "2";

    string extend = "157D924,6D";
    PNumber iP = new PNumber(a, b, c);

    string result = iP.Sqr().GetPString();
    Assert.AreEqual(extend, result);
}
[TestMethod]
public void TestSetGetBase()
{
    string a = "1283,22";
    string b = "15";
    string c = "2";

    int extend = 15;
    PNumber iP = new PNumber(a, b, c);
```

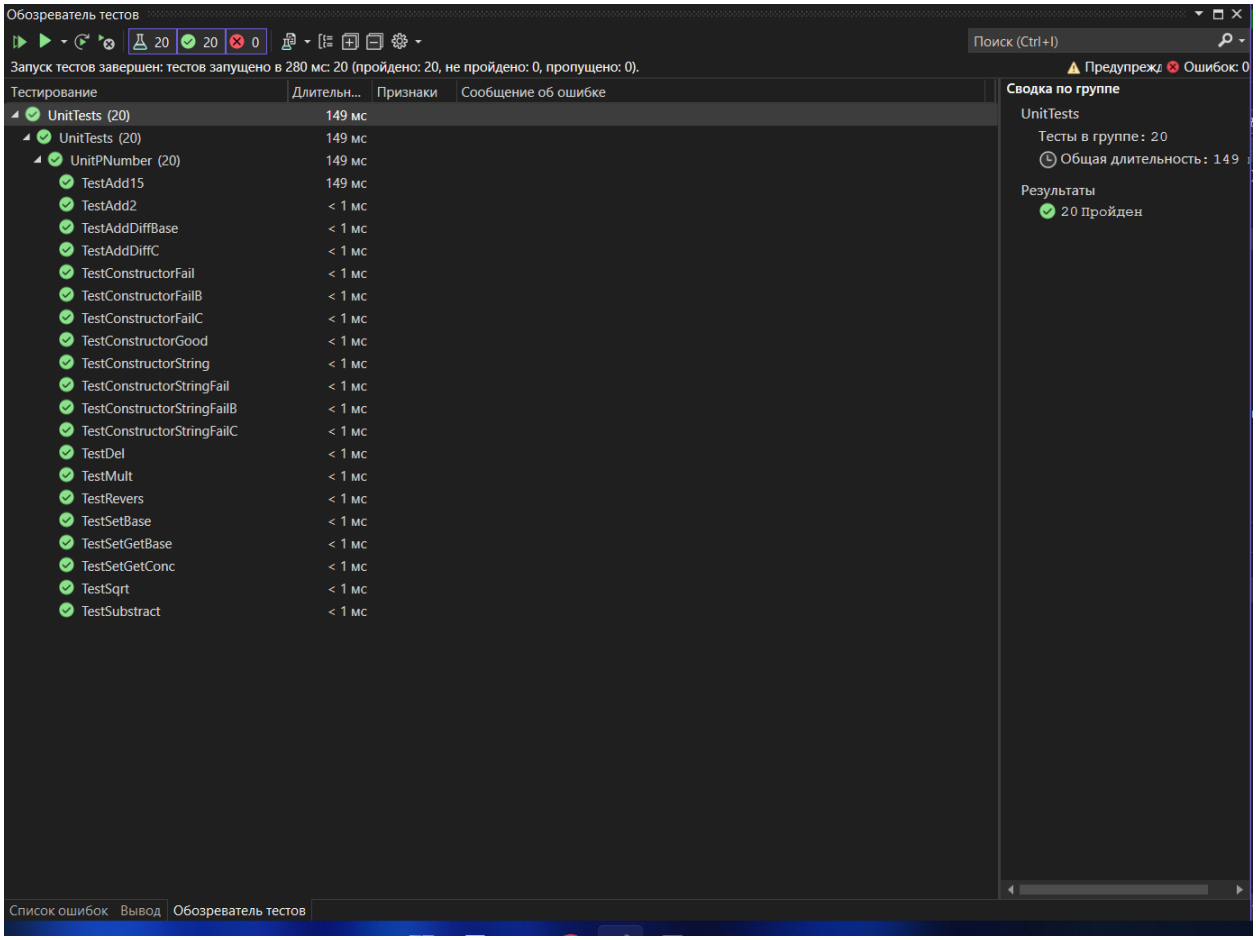
```
            iP.SetBaseNumber(2);
            int result = iP.GetBaseNumber();
            Assert.AreEqual(extend, result);
        }
        [TestMethod]
        public void TestSetGetConc()
        {
            string a = "1283,22";
            string b = "15";
            string c = "2";

            int extend = 2;
            PNumber iP = new PNumber(a, b, c);

            iP.SetCorrectnessNumber(4);
            int result = iP.GetCorrectnessNumber();
            Assert.AreEqual(extend, result);
        }
        [TestMethod]
        public void TestSetBase()
        {
            string a = "1283,22";
            string b = "15";
            string c = "2";

            int extend = 16;
            PNumber iP = new PNumber(a, b, c);

            iP.SetBaseNumber(16);
            int result = iP.GetBaseNumber();
            Assert.AreEqual(extend, result);
        }
    }
}
```

3. Результаты

3.1.Пример работы программы



3.2.Результаты тестирования



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.