

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  
«Редактор Р-ичных чисел»

Выполнил:  
Студент группы ИП-013  
Копытина Т.А.  
Работу проверил:  
ассистент кафедры ПМиК  
Агалаков А.А.

Новосибирск 2023 г.

## Содержание

1. Задание.....	3
2. Исходный код программы .....	5
2.1. Код программы .....	5
2.2. Код тестов.....	9
3. Результаты модульных тестов.....	11
4. Вывод.....	12

## 1. Задание

1. Разработать и реализовать класс TEditor «Редактор р-ичных чисел», используя класс C++.
2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования.
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

На Унифицированном языке моделирования UML (Unified Modeling Language) наш класс можно описать следующим образом:

### Редактор Р-ичныхЧисел

- строка: String
- числоЕстьНоль: Boolean
- добавитьЗнак: String
- добавитьР-ичную цифру(a: Integer): String
- добавитьНоль: String
- забойСимвола: String
- очистить: String
- конструктор
- читатьСтрокаВформатеСтроки: String (метод свойства)
- писатьСтрокаВформатеСтроки(a: String) (метод свойства)
- редактировать(a: Integer): String

Обязанность:

ввод, хранение и редактирование строкового представления р-ичных чисел

4. Класс должен отвечать за ввод и редактирование строкового представления р-ичных чисел. Значение р-ичного нуля - '0, '. Класс должен обеспечивать:
  - добавление символов, соответствующих р-ичным цифрам (р от 2 до 16);

- добавление и изменение знака;
- добавление разделителя целой и дробной частей;
- забой символа, стоящего справа (BackSpace);
- установку нулевого значения числа (Clear);
- чтение строкового представления  $p$ -ичного числа;
- запись строкового представления  $p$ -ичного числа;

5. Протестировать каждый метод класса.

## 2. Исходный код программы

### 2.1. Код программы

#### TEditor.cs

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

namespace lab8
{
    public class TEditor
    {
        public const char SPLIT = ',';
        public const string ZERO = "0,0";

        private string number;
        private int b;
        private int c;
        public TEditor()
        {
            number = ZERO;
            b = 10;
            c = 1;
        }
        public TEditor(int b, int c)
        {
            if (checkOnCountBaseAndC(b,
c))
            {
                this.b = b;
                this.c = c;
                number = ZERO;
            }
            else
            {
                number = ZERO;
                b = 10;
                c = 1;
            }
        }
        public TEditor(string number, int
b, int c)
        {
            if (check(number, b, c))
            {
                this.number = number;
                this.b = b;
                this.c = c;
            }
            else
            {
                this.number = ZERO;
                this.b = 10;
                this.c = 1;
            }
        }
        public bool IsZero()
        {
            return (number.Equals(ZERO));
        }
        public string AddSign() // 0
        {
            if (number[0].Equals('-'))
            {
                number =
number.Substring(1);
            }
            else
            {
                number = "-" + number;
            }
            return number;
        }
        public string Add(int a)
        {
            if (checkPoint(number) && c
== 1 && number[number.Length - 1] == '0')
            {
                number =
number.Substring(0, number.Length - 2);
                number += a;
                number += ",0";
            }
            return number;
        }
        public string AddZero() // 1
        {
            if (checkPoint(number) && c
== 1 && number[number.Length - 1] == '0')
            {
                number =
number.Substring(0, number.Length - 2);
                number += 0;
                number += ",0";
            }
            return number;
        }
        public string BackSpace() // 2
        {
            if (!number.Equals(ZERO) &&
number.Length > 2)
            {
                number =
number.Substring(0, number.Length - 1);
                if (number[number.Length
- 1].Equals(SPLIT))
                {
                    number += "0";
                    c++;
                }
                c--;
            }
            return number;
        }
        public string Clear() // 3
        {
            number = ZERO;
            c = 1;
            return number;
        }
    }
}
```

```

public string GetNumber() // 4
{
    return number;
}
public string SetNumber(string a)
{
    if (checkOnBase(a, b) &&
countC(a) > 0)
    {
        number = a;
        c = countC(a);
    }
    return number;
}
public String Edit(int a)
{
    switch (a)
    {
        case 0:
            AddSign();
            break;
        case 1:
            AddZero();
            break;
        case 2:
            BackSpace();
            break;
        case 3:
            Clear();
            break;
        case 4:
            GetNumber();
            break;
    }
    return number;
}
private bool check(string a, int
b, int c)
{
    if (!checkOnBase(a, b))
    {
        return false;
    }
    if (!checkOnC(a, c))
    {
        return false;
    }
    if (!checkOnSymbol(a))
    {
        return false;
    }
    if (!checkOnCountBaseAndC(b,
c))
    {
        return false;
    }
    return true;
}
private bool checkOnC(string a,
int c)
{
    if (checkPoint(a) && c > 0)

```

```

{
    string[] splitter =
a.Split(',');
    if (splitter[1].Length ==
c)
    {
        return true;
    }
    if (!checkPoint(a) && c == 0)
    {
        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 checkPoint(string n)
{
    int i;
    for (i = 0; i < n.Length &&
n[i] != ','; i++) { }
    if (i < n.Length)
    {
        return true;
    }
    return false;
}
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
checkOnCountBaseAndC(int b, int c)
    {
        if (b > 1 && b < 17 && c > 0)
        {
            return true;
        }
        return false;
    }
    public int getC()
    {
        return c;
    }

```

```

        public int getB()
        {
            return b;
        }

        private int countC(string a)
        {
            string[] split =
a.Split(",");
            return split[1].Length;
        }
    }
}

```

## **Program.cs**

```
using System;
namespace lab8
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```



## 2.2. Код тестов

### TestEditPNumber.cs

```
using
Microsoft.VisualStudio.TestTools.UnitTesting;
using lab8;

namespace TestProject1
{
    [TestClass]
    public class TestEditPNumber
    {
        [TestMethod]
        public void ConstructurZeroGood()
        {
            TEditor edit = new TEditor();
            string expected = "0,0";
            string result =
edit.GetNumber();
            Assert.AreEqual(expected,
result);
        }
        [TestMethod]
        public void ConstructurBCGood()
        {
            TEditor edit = new
TEditor(16, 0);
            string expected = "0,0";
            string result =
edit.GetNumber();
            Assert.AreEqual(expected,
result);
        }
        [TestMethod]
        public void ConstructurNumberBCGood()
        {
            TEditor edit = new
TEditor("16,16", 16, 2);
            string expected = "16,16";
            string result =
edit.GetNumber();
            Assert.AreEqual(expected,
result);
        }
        [TestMethod]
        public void IsZeroGood()
        {
            TEditor edit = new TEditor();
            bool expected = true;
            bool result = edit.IsZero();
            Assert.AreEqual(expected,
result);
        }
        [TestMethod]
        public void AddSignoGood()
        {
            TEditor edit = new
TEditor("16,16", 16, 2);
            string expected = "-16,16";
            string result =
edit.AddSign();
            Assert.AreEqual(expected,
result);
        }
        [TestMethod]
        public void AddGood()
        {
            TEditor edit = new
TEditor("16,0", 16, 1);
            string expected = "163,0";
            string result = edit.Add(3);
            Assert.AreEqual(expected,
result);
        }
        [TestMethod]
        public void AddZeroGood()
        {
            TEditor edit = new
TEditor("16,0", 16, 1);
            string expected = "160,0";
            string result =
edit.AddZero();
            Assert.AreEqual(expected,
result);
        }
        [TestMethod]
        public void BackSpaceGood()
        {
            TEditor edit = new
TEditor("16,3", 16, 1);
            string expected = "16,0";
            string result =
edit.BackSpace();
            Assert.AreEqual(expected,
result);
        }
        [TestMethod]
        public void ClearGood()
        {
            TEditor edit = new
TEditor("16,3", 16, 1);
            string expected = "0,0";
            string result = edit.Clear();
            Assert.AreEqual(expected,
result);
        }
        [TestMethod]
        public void GetNumberGood()
        {
            TEditor edit = new
TEditor("16,3", 16, 1);
            string expected = "16,3";
            string result =
edit.GetNumber();
            Assert.AreEqual(expected,
result);
        }
        [TestMethod]
        public void SetNumberGood()
        {
            TEditor edit = new
TEditor("16,3", 16, 1);
            string expected = "16,3";
            string result =
edit.SetNumber("16,3");
            Assert.AreEqual(expected,
result);
        }
    }
}
```

```

        TEditor edit = new
TEditor("16,3", 16, 1);
        string expected = "124,5324";
        edit.SetNumber("124,5324");
        string result =
edit.GetNumber();
        Assert.AreEqual(expected,
result);
    }
    [TestMethod]
    public void ConstructurBCFail()
    {
        TEditor edit = new
TEditor(17, 1);
        string expected = "0,0";
        string result =
edit.GetNumber();
        Assert.AreEqual(expected,
result);
    }
    [TestMethod]
    public void
ConstructurNumberBCFail()
    {
        TEditor edit = new
TEditor("16,3", 1, 1);
        string expected = "0,0";
        string result =
edit.GetNumber();
        Assert.AreEqual(expected,
result);
    }
    [TestMethod]
    public void IsZeroFail()
    {
        TEditor edit = new
TEditor("16,3", 16, 1);
        bool expected = false;
        bool result = edit.IsZero();
        Assert.AreEqual(expected,
result);
    }
    [TestMethod]
    public void AddSignoFail()
    {
        TEditor edit = new TEditor("-
16,3", 16, 1);
        string expected = "16,3";
        string result =
edit.AddSign();
        Assert.AreEqual(expected,
result);
    }
    [TestMethod]
    public void AddFail()

```

```

    {
        TEditor edit = new TEditor("-
16,3", 16, 1);
        string expected = "-16,3";
        string result = edit.Add(4);
        Assert.AreEqual(expected,
result);
    }
    [TestMethod]
    public void AddZeroFail()
    {
        TEditor edit = new TEditor("-
16,3", 16, 1);
        string expected = "-16,3";
        string result =
edit.AddZero();
        Assert.AreEqual(expected,
result);
    }
    [TestMethod]
    public void BackSpaceFail()
    {
        TEditor edit = new TEditor("-
16,3345", 16, 4);
        string expected = "-16,334";
        string result =
edit.BackSpace();
        Assert.AreEqual(expected,
result);
    }
    [TestMethod]
    public void ClearFail()
    {
        TEditor edit = new
TEditor("0,0", 16, 1);
        string expected = "0,0";
        string result =
edit.BackSpace();
        Assert.AreEqual(expected,
result);
    }
    [TestMethod]
    public void SetNumberFail()
    {
        TEditor edit = new
TEditor("110,0", 10, 1);
        string expected = "110,0";
        string result =
edit.SetNumber("1A,2B");
        Assert.AreEqual(expected,
result);
    }
}

```

### 3. Результаты модульных тестов

Запуск тестов завершен: тестов запущено в 288 мс: 20 (пройдено: 20, не пройдено: 0, пропущено: 0).			
Тестирование	Длительн...	Признаки	Сообщение об ошибке
TestProject1 (20)	20 мс		
TestProject1 (20)	20 мс		
TestEditPNumber (20)	20 мс		
AddFail	9 мс		
AddGood	< 1 мс		
AddSignoFail	< 1 мс		
AddSignoGood	< 1 мс		
AddZeroFail	< 1 мс		
AddZeroGood	< 1 мс		
BackSpaceFail	< 1 мс		
BackSpaceGood	< 1 мс		
ClearFail	< 1 мс		
ClearGood	< 1 мс		
ConstructurBCFail	< 1 мс		
ConstructurBCGood	< 1 мс		
ConstructurNumberBCFail	< 1 мс		
ConstructurNumberBCGood	< 1 мс		
ConstructurZeroGood	< 1 мс		
GetNumberGood	< 1 мс		
IsZeroFail	11 мс		
IsZeroGood	< 1 мс		
SetNumberFail	< 1 мс		
SetNumberGood	< 1 мс		

#### **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.