Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

Современные технологии программирования

Лабораторная работа №5

Разработка и модульное тестирование класса

Редактор P-ичных чисел

Выполнил:

студент гр.ИП-213 Дмитриев Антон Александрович
ФИО студента

«__» _____ 2025 г.

Проверил:

Преподаватель

ФИО преподавателя

«__» _____ 2025 г. Оценка_____

Новосибирск 2025 г.

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 по критерию C0.

3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

2. Исходные тексты программ.

Class.py:

```
class P_digit_number:

    DECIMAL_SEPARATOR = '.'
    ZERO_REPRESENTATION = '0.'

    def __init__(self, radix = 10):

        if (radix < 2 or radix > 16):
            raise ValueError("Основание системы счисления должно быть от 2
до 16")

        self._radix = radix
        self._string = self.ZERO_REPRESENTATION

    @property
    def radix(self):
        return self._radix

    @property
    def string(self):
        return self._string

    @string.setter
    def string(self, value):
        if not self._is_valid_string(value):
            raise ValueError("Недопустимый формат строки")
        self._string = value

    def _is_valid_string(self, s):
        if not s:
            return False

        if s.count(self.DECIMAL_SEPARATOR) > 1:
            return False

        parts = s.split(self.DECIMAL_SEPARATOR)
```

```
        integer_part = parts[0]
        fractional_part = parts[1] if len(parts) > 1 else ""

        if integer_part.startswith('-'):
            integer_part = integer_part[1:]

        valid_digits = self._get_valid_digits()

        for char in integer_part + fractional_part:
            if char not in valid_digits:
                return False

        return True

    def _get_valid_digits(self):
        digits = "0123456789ABCDEF"
        return set(digits[:self._radix])

    def is_zero(self):
        # Убираем знак и проверяем, равно ли число нулю
        clean_string = self._string.lstrip('-')
        return clean_string == self.ZERO_REPRESENTATION

    def add_sign(self):
        if self._string.startswith('-'):
            self._string = self._string[1:]
        else:
            self._string = '-' + self._string
        return self._string

    def add_radix_digit(self, digit):
        if not 0 <= digit < self._radix:
            raise ValueError(f"Цифра {digit} недопустима для системы
счисления с основанием {self._radix}")

        digits = "0123456789ABCDEF"
        char = digits[digit]

        if self.is_zero():
```

```
            self._string = char + self.DECIMAL_SEPARATOR
        else:
            if self.DECIMAL_SEPARATOR in self._string:
                parts = self._string.split(self.DECIMAL_SEPARATOR)
                integer_part = parts[0]
                fractional_part = parts[1]
                self._string = integer_part + char + self.DECIMAL_SEPARATOR +
fractional_part
            else:
                self._string += char

        return self._string

    def add_zero(self):
        return self.add_radix_digit(0)

    def backspace(self):
        if self.is_zero():
            return self._string

        # Разделяем на целую и дробную части
        if self.DECIMAL_SEPARATOR in self._string:
            parts = self._string.split(self.DECIMAL_SEPARATOR)
            integer_part = parts[0]
            fractional_part = parts[1] if len(parts) > 1 else ""

            # Если есть цифры в целой части (кроме возможного знака)
            if len(integer_part) > 1 or (len(integer_part) == 1 and not
integer_part.startswith('-')):
                # Удаляем последний символ из целой части
                new_integer_part = integer_part[:-1]

                # Если после удаления целая часть пустая или содержит только
знак
                if new_integer_part == '' or new_integer_part == '-':
                    self._string = self.ZERO_REPRESENTATION
                else:
                    self._string = new_integer_part + self.DECIMAL_SEPARATOR +
fractional_part
```

```
            else:
                # Если в целой части только одна цифра (или знак),
устанавливаем ноль
                self._string = self.ZERO_REPRESENTATION
        else:
            # Если нет разделителя (не должно происходить в нормальном
состоянии)
            if len(self._string) > 1:
                self._string = self._string[:-1]
            else:
                self._string = self.ZERO_REPRESENTATION

        return self._string

    def clear(self):
        self._string = self.ZERO_REPRESENTATION
        return self._string

    def edit(self, command):
        if command == 0:
            return self.clear()
        elif command == 1:
            return self.add_sign()
        elif command == 2:
            return self.backspace()
        elif command == 3:
            return self.add_zero()
        elif 4 <= command < 4 + self._radix:
            digit = command - 4
            return self.add_radix_digit(digit)
        else:
            raise ValueError(f"Неизвестная команда: {command}")
```

Tests.py:

```
import pytest
from lab5 import P_digit_number


class TestP_digit_number:
    """Тесты для класса P_digit_number"""

    def test_constructor_valid_radix(self):
        """Тест конструктора с допустимыми основаниями"""
        for radix in [2, 8, 10, 16]:
            editor = P_digit_number(radix)
            assert editor.radix == radix
            assert editor.string == "0."

    def test_constructor_invalid_radix(self):
        """Тест конструктора с недопустимыми основаниями"""
        with pytest.raises(ValueError):
            P_digit_number(1)
        with pytest.raises(ValueError):
            P_digit_number(17)

    def test_is_zero(self):
        """Тест метода is_zero"""
        editor = P_digit_number()
        assert editor.is_zero() == True

        editor.string = "5."
```

```
        assert editor.is_zero() == False

        editor.string = "-0."
        assert editor.is_zero() == True

    def test_add_sign(self):
        """Тест метода add_sign"""
        editor = P_digit_number()

        # Добавление знака
        result = editor.add_sign()
        assert result == "-0."
        assert editor.string == "-0."

        # Удаление знака
        result = editor.add_sign()
        assert result == "0."
        assert editor.string == "0."

    def test_add_radix_digit(self):
        """Тест метода add_radix_digit"""
        editor = P_digit_number(10)

        # Добавление цифры к нулю
        result = editor.add_radix_digit(5)
        assert result == "5."
        assert editor.string == "5."
```

```
    # Добавление второй цифры
    result = editor.add_radix_digit(3)
    assert result == "53."
    assert editor.string == "53."

def test_add_radix_digit_hex(self):
    """Тест добавления цифр в шестнадцатеричной системе"""
    editor = P_digit_number(16)

    result = editor.add_radix_digit(10)  # A
    assert result == "A."
    assert editor.string == "A."

    result = editor.add_radix_digit(15)  # F
    assert result == "AF."
    assert editor.string == "AF."

def test_add_radix_digit_invalid(self):
    """Тест добавления недопустимой цифры"""
    editor = P_digit_number(8)

    with pytest.raises(ValueError):
        editor.add_radix_digit(8)

    with pytest.raises(ValueError):
        editor.add_radix_digit(9)
```

```
def test_add_zero(self):
    """Тест метода add_zero"""
    editor = P_digit_number()

    result = editor.add_zero()
    assert result == "0."

    editor.string = "5."
    result = editor.add_zero()
    assert result == "50."

def test_backspace(self):
    """Тест метода backspace"""
    editor = P_digit_number()
    editor.string = "123."

    result = editor.backspace()
    assert result == "12."

    result = editor.backspace()
    assert result == "1."

    result = editor.backspace()
    assert result == "0."

def test_backspace_from_zero(self):
```

```
    """Тест backspace из нулевого состояния"""
    editor = P_digit_number()

    result = editor.backspace()
    assert result == "0."  # Остается нулем

def test_clear(self):
    """Тест метода clear"""
    editor = P_digit_number()
    editor.string = "12345."

    result = editor.clear()
    assert result == "0."
    assert editor.string == "0."

def test_edit_commands(self):
    """Тест метода edit с различными командами"""
    editor = P_digit_number(10)

    # Команда 0 - очистить
    editor.string = "123."
    result = editor.edit(0)
    assert result == "0."

    # Команда 1 - добавить знак
    result = editor.edit(1)
    assert result == "-0."
```

```
    # Команда 2 - забой символа
    editor.string = "12."
    result = editor.edit(2)
    assert result == "1."

    # Команда 3 - добавить ноль
    result = editor.edit(3)
    assert result == "10."

    # Команды 4-13 - добавить цифры 0-9
    result = editor.edit(4)  # 0
    assert result == "100."

    result = editor.edit(9)  # 5
    assert result == "1005."

def test_edit_invalid_command(self):
    """Тест метода edit с недопустимой командой"""
    editor = P_digit_number(10)

    with pytest.raises(ValueError):
        editor.edit(20)

def test_string_property(self):
    """Тест свойств чтения/записи строки"""
    editor = P_digit_number()
```

```
    # Тест чтения
    assert editor.string == "0."

    # Тест записи допустимой строки
    editor.string = "123.45"
    assert editor.string == "123.45"

    # Тест записи строки со знаком
    editor.string = "-123.45"
    assert editor.string == "-123.45"

def test_string_property_invalid(self):
    """Тест записи недопустимой строки"""
    editor = P_digit_number(10)

    with pytest.raises(ValueError):
        editor.string = "12A.45"  # Недопустимая цифра A для 10-чной системы

    with pytest.raises(ValueError):
        editor.string = "12.34.56"  # Много разделителей

def test_binary_system(self):
    """Тест работы с двоичной системой"""
    editor = P_digit_number(2)

    assert editor.add_radix_digit(1) == "1."
```

```
        assert editor.add_radix_digit(0) == "10."

        with pytest.raises(ValueError):
            editor.add_radix_digit(2)

    def test_hexadecimal_system(self):
        """Тест работы с шестнадцатеричной системой"""
        editor = P_digit_number(16)

        editor.add_radix_digit(10)  # A
        editor.add_radix_digit(11)  # B
        editor.add_radix_digit(12)  # C

        assert editor.string == "ABC."

        # Проверка допустимости всех цифр
        valid_digits = set("0123456789ABCDEF")
        for i in range(16):
            editor.clear()
            editor.add_radix_digit(i)
            assert editor.string[0] in valid_digits

if __name__ == "__main__":
    pytest.main()
```

3. Результат тестирования методов класса Matrix.

1. Тестирование конструктора

1.1 Допустимые основания систем счисления

Входные данные (radix)	Ожидаемый результат
2	radix = 2, string = "0."
8	radix = 8, string = "0."
10	radix = 10, string = "0."
16	radix = 16, string = "0."

1.2 Недопустимые основания систем счисления

Входные данные (radix)	Ожидаемый результат
1	ValueError
17	ValueError

2. Тестирование метода is_zero()

Начальное состояние	Ожидаемый результат
"0."	True
"5."	False
"-0."	True
"123."	False

3. Тестирование метода add_sign()

Начальное состояние	Ожидаемый результат
"0."	"-0."
"-0."	"0."
"5."	"-5."
"-5."	"5."

4. Тестирование метода add_radix_digit()

4.1 Десятичная система (radix = 10)

Начальное состояние	Входная цифра	Ожидаемый результат
"0."	5	"5."
"5."	3	"53."
"53."	0	"530."

4.2 Шестнадцатеричная система (radix = 16)

Начальное состояние	Входная цифра	Ожидаемый результат
"0."	10	"A."
"A."	15	"AF."
"AF."	12	"AFC."

4.3 Недопустимые цифры для восьмеричной системы (radix = 8)

Начальное состояние	Входная цифра	Ожидаемый результат
"0."	8	ValueError
"0."	9	ValueError

5. Тестирование метода add_zero()

Начальное состояние	Ожидаемый результат
"0."	"0."
"5."	"50."
"123."	"1230."

6. Тестирование метода backspace()

Начальное состояние	Ожидаемый результат
"123."	"12."
"12."	"1."
"1."	"0."
"0."	"0."

7. Тестирование метода clear()

Начальное состояние	Ожидаемый результат
"12345."	"0."
"-567."	"0."
"0."	"0."

8. Тестирование метода edit()

8.1 Десятичная система (radix = 10)

Начальное состояние	Команда	Ожидаемый результат
"123."	0	"0."
"0."	1	"-0."
"-0."	1	"0."
"12."	2	"1."
"1."	3	"10."
"10."	4	"100."
"100."	9	"1005."

8.2 Недопустимые команды

Начальное состояние	Команда	Ожидаемый результат
"0."	20	ValueError

9. Тестирование свойства string

9.1 Запись допустимых строк

Начальное состояние	Новая строка	Ожидаемый результат
"0."	"123.45"	"123.45"
"0."	"-123.45"	"-123.45"

9.2 Запись недопустимых строк (radix = 10)

Начальное состояние	Новая строка	Ожидаемый результат
"0."	"12A.45"	ValueError
"0."	"12.34.56"	ValueError

10. Тестирование двоичной системы (radix = 2)

Начальное состояние	Входная цифра	Ожидаемый результат
"0."	1	"1."
"1."	0	"10."
"0."	2	ValueError

11. Тестирование шестнадцатеричной системы (radix = 16)

Начальное состояние	Входная цифра	Ожидаемый результат
"0."	10	"A."
"A."	11	"AB."
"AB."	12	"ABC."
"ABC."	13	"ABCD."
"ABCD."	14	"ABCDE."
"ABCDE."	15	"ABCDEF."