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

Лабораторная работа №9

Выполнил:
студенты 4 курса
группы ИП-216
Андрущенко Ф.А.
Литвинов А. Е.
Русецкий А. С.

Проверил:
преподаватель кафедры ПМиК
Агалаков Антон Александрович

Новосибирск, 2025 г.

Задание

1. Реализовать тип «полином», в соответствии с приведенной ниже спецификацией
2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования Visual Studio
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций

Спецификация абстрактного типа данных «Полином».
ADT TPoly
Данные
Полиномы Tpoly - это неизменяемые полиномы с целыми коэффициентами.
Операции
Операции могут вызываться только объектом «полином» (тип TPoly), указатель на который передаётся в них по умолчанию. При описании операций этот объект в разделе «Вход» не указывается.

Таблица 1. Описание операций на ADT TPoly

Наименование операции	Описание
Конструктор	
Начальные значения:	Коэффициент (c) и степень (n) одночленного полинома
Процесс:	Создаёт одночленный полином с коэффициентом (c) и степенью (n), или ноль-полином, если коэффициент (c) равен 0 и возвращает указатель на него. Например: Конструктор(6,3) = 6x^3 Конструктор(3,0) = 3 Конструктор() = 0
Выход:	Нет.
Постусловия:	Объект инициализирован.
Степень	

Вход:	Нет.
Предусловия:	Нет.
Процесс:	Отыскивает степень n полинома, т.е. наибольшую степень при ненулевом коэффициенте (c). Степень нулевого полинома равна 0. Например: a = (x^2+1), a.Степень = 2 a = (17), a. Степень = 0
Выход:	n - целое число - степень полинома.
Постусловия:	Нет.
Коэффициент	
Вход:	n - целое число - степень полинома.
Предусловия:	Полином – не нулевой.
Процесс:	Отыскивает коэффициент (c) при члене полинома со степенью n (c*x^n). Возвращает коэффициент (c) найденного члена или 0, если n больше степени полинома. Например: p = (x^3+2x+1), p.Coeff (4) = 0 p = (x^3+2x+1), p.Coeff (1) = 2
Выход:	Целое число.
Постусловия:	Нет.
Очистить (Clear)	
Вход:	Нет.
Предусловия:	Нет
Процесс:	Удаляет члены полинома.
Выход:	Нет.
Постусловия:	this – нуль-полином.
Сложить	
Вход:	q - полином.
Предусловия:	Нет
Процесс:	Создаёт полином, являющийся результатом сложения полинома с полиномом q и возвращает его.
Выход:	Полином.
Постусловия:	Нет.

Умножить	
Вход:	q - полином.
Предусловия:	Нет.
Процесс:	Создаёт полином, являющийся результатом умножения полинома на полином q и возвращает его.
Выход:	Полином.
Постусловия:	Нет.
Вычесть	
Вход:	q - полином.
Предусловия:	Нет.
Процесс:	Создаёт полином, являющийся результатом вычитания из полинома полинома q, и возвращает его.
Выход:	Полином.
Постусловия:	Нет.
Минус	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт полином, являющийся разностью ноль-полинома, и полинома и возвращает его.
Выход:	Полином.
Постусловия:	Нет.
Равно	
Вход:	q - полином.
Предусловия:	Нет.
Процесс:	Сравнивает полином с полиномом q на равенство. Возвращает значение True, если полиномы равны, т.е. имеют одинаковые коэффициенты при соответствующих членах, и значение False - в противном случае.
Выход:	Булевское значение.
Постусловия:	Нет.
Дифференцировать	

Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт полином, являющийся производной полинома и возвращает его. Например: a = (x^3+7x+5), a.Дифференцировать = $3x^2$+7
Выход:	Полином.
Постусловия:	Нет.
Вычислить	
Вход:	x – действительное число.
Предусловия:	Нет.
Процесс:	Вычисляет значение полинома в точке x и возвращает его. Например: a = (x^2+3x), a.Вычислить(2) = 10
Выход:	Действительное число.
Постусловия:	Нет.
Элемент	
Вход:	i - целое число - номер члена полинома.
Предусловия:	Нет.
Процесс:	Обеспечивает доступ к члену полинома с индексом i для чтения его коэффициента (c) и степени (n) так, что если изменять i от 0 до количества членов в полиноме минус один, то можно просмотреть все члены полинома.
Выход:	Коэффициент – целое число, степень – целое число.
Постусловия:	Полином не модифицируется.

Исходный код программы

```
class TMember:
    """Класс для представления одночлена"""

    def __init__(self, c: int = 0, n: int = 0):
        """
        Конструктор одночлена
        c - коэффициент, n - степень
        """
        self._coeff = c
        self._degree = n if c != 0 else 0

    def get_degree(self) -> int:
        """ЧитатьСтепень - возвращает степень одночлена"""
        return self._degree

    def set_degree(self, n: int):
        """ПисатьСтепень - устанавливает степень одночлена"""
        self._degree = n
        if self._coeff == 0:
            self._degree = 0

    def get_coeff(self) -> int:
        """ЧитатьКоэффициент - возвращает коэффициент одночлена"""
        return self._coeff

    def set_coeff(self, c: int):
        """ПисатьКоэффициент - устанавливает коэффициент одночлена"""
        self._coeff = c
        if c == 0:
            self._degree = 0

    def __eq__(self, other: 'TMember') -> bool:
        """Равно - сравнение одночленов"""
        if not isinstance(other, TMember):
            return False
        return self._coeff == other._coeff and self._degree == other._degree

    def differentiate(self) -> 'TMember':
        """Дифференцировать - производная одночлена"""
        if self._degree == 0:
            return TMember(0, 0)
        return TMember(self._coeff * self._degree, self._degree - 1)

    def evaluate(self, x: float) -> float:
        """Вычислить - вычисление значения одночлена в точке x"""
        return self._coeff * (x ** self._degree)

    def to_string(self) -> str:
        """ОдночленВСтроку - строковое представление одночлена"""
```

```
        if self._coeff == 0:
            return "0"

        result = ""

        if self._coeff != 1 or self._degree == 0:
            if self._coeff == -1 and self._degree != 0:
                result += "-"
            else:
                result += str(self._coeff)

        if self._degree > 0:
            result += "x"
            if self._degree > 1:
                result += f"^{self._degree}"

        return result

    def __str__(self):
        return self.to_string()


class TPoly:
    """Класс для представления полинома"""

    def __init__(self, c: int = 0, n: int = 0):
        """
        Конструктор полинома
        c - коэффициент, n - степень для создания одночленного полинома
        """
        self._members = []
        if c != 0:
            self._members.append(TMember(c, n))
        self._normalize()

    def _normalize(self):
        """Приведение полинома к нормализованному виду"""
        if not self._members:
            return

        self._members.sort(key=lambda m: m.get_degree(), reverse=True)

        i = 0
        while i < len(self._members) - 1:
            current = self._members[i]
            next_member = self._members[i + 1]

            if current.get_degree() == next_member.get_degree():
                new_coeff = current.get_coeff() + next_member.get_coeff()
                if new_coeff != 0:
                    self._members[i] = TMember(new_coeff, current.get_degree())
                    del self._members[i + 1]
```

```
            else:
                del self._members[i]
                del self._members[i]
        else:
            i += 1

    self._members = [m for m in self._members if m.get_coeff() != 0]

def get_degree(self) -> int:
    """Степень - возвращает степень полинома"""
    if not self._members:
        return 0
    return max(m.get_degree() for m in self._members)

def get_coeff(self, n: int) -> int:
    """Коэффициент - возвращает коэффициент при степени n"""
    if not self._members:
        return 0
    for member in self._members:
        if member.get_degree() == n:
            return member.get_coeff()
    return 0

def clear(self):
    """Очистить - удаляет все члены полинома"""
    self._members = []

def add(self, other: 'TPoly') -> 'TPoly':
    """Сложить - сложение полиномов"""
    result = TPoly()
    result._members = self._members.copy()
    for member in other._members:
        result._members.append(TMember(member.get_coeff(), member.get_degree()))
    result._normalize()
    return result

def multiply(self, other: 'TPoly') -> 'TPoly':
    """Умножить - умножение полиномов"""
    result = TPoly()
    for m1 in self._members:
        for m2 in other._members:
            new_coeff = m1.get_coeff() * m2.get_coeff()
            new_degree = m1.get_degree() + m2.get_degree()
            result._members.append(TMember(new_coeff, new_degree))
    result._normalize()
    return result

def subtract(self, other: 'TPoly') -> 'TPoly':
    """Вычесть - вычитание полиномов"""
    return self.add(other.negate())

def negate(self) -> 'TPoly':
```

```
    """Минус - унарный минус"""
    result = TPoly()
    for member in self._members:
        result._members.append(TMember(-member.get_coeff(), member.get_degree()))
    result._normalize()
    return result

def __eq__(self, other: 'TPoly') -> bool:
    """Равно - сравнение полиномов"""
    if not isinstance(other, TPoly):
        return False
    if len(self._members) != len(other._members):
        return False
    for m1, m2 in zip(self._members, other._members):
        if m1 != m2:
            return False
    return True

def differentiate(self) -> 'TPoly':
    """Дифференцировать - производная полинома"""
    result = TPoly()
    for member in self._members:
        if member.get_degree() > 0:
            derivative = member.differentiate()
            result._members.append(derivative)
    result._normalize()
    return result

def evaluate(self, x: float) -> float:
    """Вычислить - вычисление значения полинома в точке x"""
    result = 0.0
    for member in self._members:
        result += member.evaluate(x)
    return result

def get_member(self, i: int) -> tuple:
    """Элемент - доступ к члену полинома по индексу"""
    if i < 0 or i >= len(self._members):
        raise IndexError("Индекс выходит за границы")
    member = self._members[i]
    return member.get_coeff(), member.get_degree()

def to_string(self) -> str:
    """Строковое представление полинома"""
    if not self._members:
        return "0"

    result = ""
    for i, member in enumerate(self._members):
        member_str = member.to_string()
        if i > 0:
            if member_str.startswith('-'):
```

```
                    result += " - " + member_str[1:]
                else:
                    result += " + " + member_str
            else:
                result += member_str

        return result

    def __str__(self):
        return self.to_string()

    def __add__(self, other):
        return self.add(other)

    def __sub__(self, other):
        return self.subtract(other)

    def __mul__(self, other):
        return self.multiply(other)

    def __neg__(self):
        return self.negate()


if __name__ == "__main__":
    print("Тестирование класса TMember:")
    m1 = TMember(3, 2)
    m2 = TMember(-1, 3)
    m3 = TMember(0, 5)
    print(f"m1 = {m1}")
    print(f"m2 = {m2}")
    print(f"m3 = {m3}")

    print(f"Степень m1: {m1.get_degree()}")
    print(f"Коэффициент m1: {m1.get_coeff()}")
    m1.set_coeff(5)
    m1.set_degree(4)
    print(f"После изменения: {m1}")

    m4 = TMember(4, 3)
    print(f"Производная {m4}: {m4.differentiate()}")

    print(f"m4(2) = {m4.evaluate(2)}")

    print("\nТестирование класса TPoly:")
    p1 = TPoly(2, 3)
    p2 = TPoly(3, 1)
    p3 = TPoly(5, 0)
    p4 = TPoly()
    print(f"p1 = {p1}")
    print(f"p2 = {p2}")
    print(f"p3 = {p3}")
```

```
print(f"p4 = {p4}")

p5 = p1.add(p2)
print(f"p1 + p2 = {p5}")

p6 = TPoly(1, 1)
p7 = TPoly(1, 1)
p8 = p6.multiply(p7)
print(f"x * x = {p8}")

p9 = p5.subtract(p1)
print(f"(2x^3 + 3x) - 2x^3 = {p9}")

p10 = p1.negate()
print(f"-p1 = {p10}")

p11 = TPoly(3, 1)
print(f"p2 == p11: {p2 == p11}")

p12 = TPoly(1, 3).add(TPoly(7, 1)).add(TPoly(5, 0))
print(f"Производная {p12}: {p12.differentiate()}")

p13 = TPoly(1, 2).add(TPoly(3, 1))
print(f"p13(2) = {p13.evaluate(2)}")

print("Члены полинома p12:")
for i in range(len(p12._members)):
    coeff, degree = p12.get_member(i)
    print(f" Член {i}: коэффициент={coeff}, степень={degree}")
```

Модульные тесты для тестирования класса

```
import unittest
from tpoly import TMember, TPoly


class TestTMember(unittest.TestCase):
    """Тесты для класса TMember"""

    def test_initialization(self):
        """Тест инициализации одночлена"""
        m1 = TMember(3, 2)
        self.assertEqual(m1.get_coeff(), 3)
        self.assertEqual(m1.get_degree(), 2)

        m2 = TMember(0, 5)
        self.assertEqual(m2.get_coeff(), 0)
        self.assertEqual(m2.get_degree(), 0)

    def test_setters(self):
        """Тест установки значений"""
        m = TMember(3, 2)
        m.set_coeff(5)
        m.set_degree(4)
        self.assertEqual(m.get_coeff(), 5)
        self.assertEqual(m.get_degree(), 4)

        m.set_coeff(0)
        self.assertEqual(m.get_degree(), 0)

    def test_equality(self):
        """Тест сравнения одночленов"""
        m1 = TMember(3, 2)
        m2 = TMember(3, 2)
        m3 = TMember(3, 3)
        self.assertTrue(m1 == m2)
        self.assertFalse(m1 == m3)

    def test_differentiate(self):
        """Тест дифференцирования одночлена"""
        m1 = TMember(4, 3)
        dm1 = m1.differentiate()
        self.assertEqual(dm1.get_coeff(), 12)
        self.assertEqual(dm1.get_degree(), 2)

        m2 = TMember(5, 0)
        dm2 = m2.differentiate()
        self.assertEqual(dm2.get_coeff(), 0)

    def test_evaluate(self):
        """Тест вычисления одночлена"""
```

```
        m = TMember(4, 3)
        self.assertEqual(m.evaluate(2), 32)

    def test_to_string(self):
        """Тест строкового представления"""
        self.assertEqual(TMember(3, 2).to_string(), "3x^2")
        self.assertEqual(TMember(-1, 3).to_string(), "-x^3")
        self.assertEqual(TMember(0, 5).to_string(), "0")
        self.assertEqual(TMember(5, 0).to_string(), "5")


class TestTPoly(unittest.TestCase):
    """Тесты для класса TPoly"""

    def test_initialization(self):
        """Тест инициализации полинома"""
        p1 = TPoly(2, 3)
        self.assertEqual(p1.get_degree(), 3)
        self.assertEqual(p1.get_coeff(3), 2)

        p2 = TPoly()
        self.assertEqual(p2.get_degree(), 0)

    def test_add(self):
        """Тест сложения полиномов"""
        p1 = TPoly(2, 3)
        p2 = TPoly(3, 1)
        p3 = p1.add(p2)
        self.assertEqual(p3.get_coeff(3), 2)
        self.assertEqual(p3.get_coeff(1), 3)

    def test_subtract(self):
        """Тест вычитания полиномов"""
        p1 = TPoly(5, 2)
        p2 = TPoly(3, 2)
        p3 = p1.subtract(p2)
        self.assertEqual(p3.get_coeff(2), 2)

    def test_multiply(self):
        """Тест умножения полиномов"""
        p1 = TPoly(1, 1)
        p2 = TPoly(1, 1)
        p3 = p1.multiply(p2)
        self.assertEqual(p3.get_degree(), 2)
        self.assertEqual(p3.get_coeff(2), 1)

    def test_negate(self):
        """Тест унарного минуса"""
        p1 = TPoly(2, 3)
        p2 = p1.negate()
        self.assertEqual(p2.get_coeff(3), -2)
```

```
    def test_equality(self):
        """Тест сравнения полиномов"""
        p1 = TPoly(3, 1)
        p2 = TPoly(3, 1)
        p3 = TPoly(2, 1)
        self.assertTrue(p1 == p2)
        self.assertFalse(p1 == p3)

    def test_differentiate(self):
        """Тест дифференцирования полинома"""
        p1 = TPoly(1, 3).add(TPoly(7, 1)).add(TPoly(5, 0))
        dp1 = p1.differentiate()
        self.assertEqual(dp1.get_coeff(2), 3)
        self.assertEqual(dp1.get_coeff(0), 7)

    def test_evaluate(self):
        """Тест вычисления полинома"""
        p1 = TPoly(1, 2).add(TPoly(3, 1))
        self.assertEqual(p1.evaluate(2), 10)

    def test_clear(self):
        """Тест очистки полинома"""
        p = TPoly(5, 3)
        p.clear()
        self.assertEqual(p.get_degree(), 0)

    def test_get_member(self):
        """Тест доступа к членам полинома"""
        p = TPoly(1, 3).add(TPoly(7, 1))
        coeff, degree = p.get_member(0)
        self.assertEqual(degree, 3)
        self.assertEqual(coeff, 1)

        with self.assertRaises(IndexError):
            p.get_member(10)


if __name__ == "__main__":
    from rich import print
    from rich.panel import Panel
    from rich.console import Console
    from unittest import TextTestRunner, TestResult, TestSuite

    console = Console()

    class RichTestResult(TestResult):
        def __init__(self, stream, descriptions, verbosity):
            super().__init__(stream, descriptions, verbosity)

        def startTest(self, test):
            super().startTest(test)
            console.print(f"[cyan]Running:[/cyan] {test._testMethodName}")
```

```
        def addSuccess(self, test):
            super().addSuccess(test)
            console.print(f"[green]✓ PASS:[/green] {test._testMethodName}")

        def addFailure(self, test, err):
            super().addFailure(test, err)
            console.print(f"[red]✗ FAIL:[/red] {test._testMethodName}")

        def addError(self, test, err):
            super().addError(test, err)
            console.print(f"[magenta]💥 ERROR:[/magenta] {test._testMethodName}")

    loader = unittest.defaultTestLoader
    suite = TestSuite()
    suite.addTests(loader.loadTestsFromTestCase(TestTPoly))

    runner = TextTestRunner(resultclass=RichTestResult, verbosity=0)
    result = runner.run(suite)

    console.print(
        Panel.fit(
            f"[green]Passed: {result.testsRun - len(result.failures) -
len(result.errors)}[/green]\n"
            f"[red]Failed: {len(result.failures)}[/red]\n"
            f"[magenta]Errors: {len(result.errors)}[/magenta]\n"
            f"[yellow]Total: {result.testsRun}[/yellow]",
            title="Test Results",
        )
    )
```

Результат тестирования методов класса

```
❯ python tpoly.py
Тестирование класса TMember:
m1 = 3x^2
m2 = -x^3
m3 = 0
Степень m1: 2
Коэффициент m1: 3
После изменения: 5x^4
Производная 4x^3: 12x^2
m4(2) = 32

Тестирование класса TPoly:
p1 = 2x^3
p2 = 3x
p3 = 5
p4 = 0
p1 + p2 = 2x^3 + 3x
x * x = x^2
(2x^3 + 3x) - 2x^3 = 3x
-p1 = -2x^3
p2 == p11: True
Производная x^3 + 7x + 5: 3x^2 + 7
p13(2) = 10.0
Члены полинома p12:
 Член 0: коэффициент=1, степень=3
 Член 1: коэффициент=7, степень=1
 Член 2: коэффициент=5, степень=0
```