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 №9

Практическая работа. Абстрактный тип данных (ADT) «Полином»

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. 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.

2. Исходные тексты программ.

Class.py:

```
from typing import List, Tuple, Optional
from collections import defaultdict
import math

class TMember:
    """Класс для представления одночлена (члена полинома)"""

    def __init__(self, coeff: int = 0, degree: int = 0):
        """
        Конструктор одночлена
        Args:
            coeff: коэффициент одночлена
            degree: степень одночлена
        """
        self._coeff = coeff
        self._degree = degree if coeff != 0 else 0

    @property
    def coeff(self) -> int:
        """Чтение коэффициента"""
        return self._coeff

    @property
    def degree(self) -> int:
        """Чтение степени"""
        return self._degree

    @coeff.setter
    def coeff(self, value: int) -> None:
        """Запись коэффициента"""
        self._coeff = value
        if value == 0:
            self._degree = 0

    @degree.setter
    def degree(self, value: int) -> None:
        """Запись степени"""
        self._degree = value
        if self._coeff == 0:
            self._degree = 0

    def __eq__(self, other: object) -> bool:
```

```
    """Сравнение одночленов на равенство"""
    if not isinstance(other, TMember):
        return False
    return self._coeff == other._coeff and self._degree == other._degree

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

def evaluate(self, x: float) -> float:
    """Вычисление значения одночлена в точке x"""
    if self._degree == 0:
        return float(self._coeff)
    return float(self._coeff) * (x ** self._degree)

def to_string(self) -> str:
    """Строковое представление одночлена"""
    if self._coeff == 0:
        return "0"

    result = ""

    # Коэффициент
    if self._coeff != 1 and self._coeff != -1:
        result += str(self._coeff)
    elif self._coeff == -1:
        result += "-"

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

    # Если степень 0, а коэффициент не отобразился
    if self._degree == 0 and (self._coeff == 1 or self._coeff == -1):
        result += str(abs(self._coeff))

    return result

def __str__(self) -> str:
    return self.to_string()
```

```
    def __repr__(self) -> str:
        return f"TMember(coeff={self._coeff}, degree={self._degree})"

    def is_zero(self) -> bool:
        """Проверка, является ли одночлен нулевым"""
        return self._coeff == 0


class TPoly:
    """Класс для представления полинома с целыми коэффициентами"""

    def __init__(self, coeff: int = 0, degree: int = 0):
        """
        Конструктор полинома
        Args:
            coeff: коэффициент для одночленного полинома
            degree: степень для одночленного полинома
        """
        self._members: List[TMember] = []
        if coeff != 0:
            self._members.append(TMember(coeff, degree))
        self._normalize()

    def _normalize(self) -> None:
        """Нормализация полинома: сортировка по убыванию степени и удаление нулевых членов"""
        if not self._members:
            return

        # Объединение членов с одинаковой степенью
        coeff_dict = defaultdict(int)
        for member in self._members:
            if member.coeff != 0:
                coeff_dict[member.degree] += member.coeff

        # Создание нового списка членов
        self._members = []
        for degree in sorted(coeff_dict.keys(), reverse=True):
            coeff = coeff_dict[degree]
            if coeff != 0:
                self._members.append(TMember(coeff, degree))

    def _copy(self) -> 'TPoly':
        """Создание копии полинома"""
        new_poly = TPoly()
```

```
    new_poly._members = [TMember(m.coeff, m.degree) for m in self._members]
    return new_poly

@property
def degree(self) -> int:
    """Степень полинома"""
    if not self._members:
        return 0
    return self._members[0].degree

def coeff(self, n: int) -> int:
    """
    Коэффициент при степени n
    Args:
        n: степень
    Returns:
        Коэффициент при x^n или 0, если такой степени нет
    """
    for member in self._members:
        if member.degree == n:
            return member.coeff
    return 0

def clear(self) -> None:
    """Очистка полинома (превращение в нуль-полином)"""
    self._members = []

def add(self, other: 'TPoly') -> 'TPoly':
    """Сложение полиномов"""
    result = self._copy()

    # Добавляем члены второго полинома
    for member in other._members:
        result._members.append(TMember(member.coeff, member.degree))

    result._normalize()
    return result

def multiply(self, other: 'TPoly') -> 'TPoly':
    """Умножение полиномов"""
    result = TPoly()

    # Перемножаем каждый член первого полинома с каждым членом второго
    for m1 in self._members:
        for m2 in other._members:
```

```
                new_coeff = m1.coeff * m2.coeff
                new_degree = m1.degree + m2.degree
                result._members.append(TMember(new_coeff, new_degree))

        result._normalize()
        return result

    def subtract(self, other: 'TPoly') -> 'TPoly':
        """Вычитание полиномов"""
        result = self._copy()

        # Добавляем члены второго полинома с обратными знаками
        for member in other._members:
            result._members.append(TMember(-member.coeff, member.degree))

        result._normalize()
        return result

    def negate(self) -> 'TPoly':
        """Унарный минус (противоположный полином)"""
        result = TPoly()
        result._members = [TMember(-member.coeff, member.degree) for member in
self._members]
        result._normalize()
        return result

    def __eq__(self, other: object) -> bool:
        """Сравнение полиномов на равенство"""
        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.coeff != m2.coeff or m1.degree != m2.degree:
                return False

        return True

    def differentiate(self) -> 'TPoly':
        """Дифференцирование полинома"""
        result = TPoly()

        for member in self._members:
```

```
            if member.degree > 0:
                deriv = member.differentiate()
                if deriv.coeff != 0:
                    result._members.append(deriv)

        result._normalize()
        return result

    def evaluate(self, x: float) -> float:
        """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) -> Optional[Tuple[int, int]]:
        """
        Доступ к i-му члену полинома
        Args:
            i: индекс члена (начиная с 0)
        Returns:
            Кортеж (коэффициент, степень) или None, если индекс вне диапазона
        """
        if 0 <= i < len(self._members):
            member = self._members[i]
            return (member.coeff, member.degree)
        return None

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

        terms = []
        for member in self._members:
            term_str = member.to_string()
            terms.append(term_str)

        # Собираем все члены в строку
        result = terms[0]
        for term in terms[1:]:
            if term.startswith("-"):
                result += f" - {term[1:]}"
            else:
                result += f" + {term}"
```

```
    return result

def __repr__(self) -> str:
    return f"TPoly({str(self)})"

def is_zero(self) -> bool:
    """Проверка, является ли полином нулевым"""
    return len(self._members) == 0

def to_polynomial_string(self) -> str:
    """Альтернативное строковое представление полинома"""
    return str(self)

def __add__(self, other: 'TPoly') -> 'TPoly':
    """Перегрузка оператора +"""
    return self.add(other)

def __sub__(self, other: 'TPoly') -> 'TPoly':
    """Перегрузка оператора -"""
    return self.subtract(other)

def __mul__(self, other: 'TPoly') -> 'TPoly':
    """Перегрузка оператора *"""
    return self.multiply(other)

def __neg__(self) -> 'TPoly':
    """Перегрузка унарного минуса"""
    return self.negate()
```

Tests.py:

```
import pytest
import math
from lab9 import TPoly, TMember


class TestTMember:
    """Тесты для класса TMember (одночлен)"""

    def test_constructor(self):
        """Тест конструктора одночлена"""
        # Одночленный полином
        m1 = TMember(6, 3)
        assert m1.coeff == 6
        assert m1.degree == 3

        # Константа
        m2 = TMember(3, 0)
        assert m2.coeff == 3
        assert m2.degree == 0

        # Нуль-полином
        m3 = TMember()
        assert m3.coeff == 0
        assert m3.degree == 0

        # Нуль-полином при нулевом коэффициенте
        m4 = TMember(0, 5)
        assert m4.coeff == 0
        assert m4.degree == 0
```

```
def test_equality(self):
    """Тест сравнения одночленов"""
    m1 = TMember(5, 3)
    m2 = TMember(5, 3)
    m3 = TMember(5, 2)
    m4 = TMember(4, 3)

    assert m1 == m2
    assert not (m1 == m3)
    assert not (m1 == m4)
    assert not (m3 == m4)

def test_differentiate(self):
    """Тест дифференцирования одночлена"""
    # x^3 -> 3x^2
    m1 = TMember(1, 3)
    deriv1 = m1.differentiate()
    assert deriv1.coeff == 3
    assert deriv1.degree == 2

    # 5x^2 -> 10x
    m2 = TMember(5, 2)
    deriv2 = m2.differentiate()
    assert deriv2.coeff == 10
    assert deriv2.degree == 1

    # Константа -> 0
    m3 = TMember(7, 0)
    deriv3 = m3.differentiate()
    assert deriv3.coeff == 0
```

```
    assert deriv3.degree == 0

    # 2x -> 2
    m4 = TMember(2, 1)
    deriv4 = m4.differentiate()
    assert deriv4.coeff == 2
    assert deriv4.degree == 0

def test_evaluate(self):
    """Тест вычисления значения одночлена"""
    m1 = TMember(3, 2)  # 3x^2
    assert abs(m1.evaluate(2) - 12) < 0.0001  # 3*4 = 12
    assert abs(m1.evaluate(0) - 0) < 0.0001
    assert abs(m1.evaluate(-1) - 3) < 0.0001  # 3*1 = 3

    m2 = TMember(5, 0)  # 5
    assert abs(m2.evaluate(10) - 5) < 0.0001
    assert abs(m2.evaluate(-5) - 5) < 0.0001

    m3 = TMember(-2, 3)  # -2x^3
    assert abs(m3.evaluate(3) + 54) < 0.0001  # -2*27 = -54

def test_to_string(self):
    """Тест строкового представления"""
    assert str(TMember(1, 3)) == "x^3"
    assert str(TMember(-1, 3)) == "-x^3"
    assert str(TMember(5, 3)) == "5x^3"
    assert str(TMember(-5, 3)) == "-5x^3"
    assert str(TMember(1, 1)) == "x"
    assert str(TMember(-1, 1)) == "-x"
```

```
        assert str(TMember(7, 0)) == "7"
        assert str(TMember(-7, 0)) == "-7"
        assert str(TMember(0, 5)) == "0"
        assert str(TMember(1, 0)) == "1"
        assert str(TMember(-1, 0)) == "-1"

    def test_setters(self):
        """Тест сеттеров"""
        m = TMember(2, 3)

        m.coeff = 5
        assert m.coeff == 5
        assert m.degree == 3

        m.degree = 4
        assert m.coeff == 5
        assert m.degree == 4

        # При установке нулевого коэффициента степень становится 0
        m.coeff = 0
        assert m.coeff == 0
        assert m.degree == 0


class TestTPoly:
    """Тесты для класса TPoly (полином)"""

    def test_constructor(self):
        """Тест конструктора полинома"""
        # Одночленный полином
```

```
    p1 = TPoly(6, 3)
    assert p1.degree == 3
    assert p1.coeff(3) == 6

    # Константа
    p2 = TPoly(3, 0)
    assert p2.degree == 0
    assert p2.coeff(0) == 3

    # Нуль-полином
    p3 = TPoly()
    assert p3.degree == 0
    assert p3.is_zero()

    # Нуль-полином при нулевом коэффициенте
    p4 = TPoly(0, 5)
    assert p4.degree == 0
    assert p4.is_zero()

def test_degree(self):
    """Тест получения степени полинома"""
    # Создаем полином x^2 + 1
    p1 = TPoly(1, 2)  # x^2
    p1 = p1.add(TPoly(1, 0))  # + 1
    assert p1.degree == 2

    p2 = TPoly(17, 0)  # 17
    assert p2.degree == 0

    p3 = TPoly()  # 0
```

```
    assert p3.degree == 0

def test_coeff(self):
    """Тест получения коэффициента"""
    # Создаем полином x^3 + 2x + 1
    p = TPoly(1, 3)  # x^3
    p = p.add(TPoly(2, 1))  # + 2x
    p = p.add(TPoly(1, 0))  # + 1

    assert p.coeff(4) == 0  # Степени 4 нет
    assert p.coeff(3) == 1  # При x^3
    assert p.coeff(1) == 2  # При x
    assert p.coeff(0) == 1  # При константе
    assert p.coeff(2) == 0  # Степени 2 нет

def test_clear(self):
    """Тест очистки полинома"""
    p = TPoly(1, 2)
    p = p.add(TPoly(3, 1))
    assert not p.is_zero()

    p.clear()
    assert p.is_zero()
    assert p.degree == 0

def test_addition(self):
    """Тест сложения полиномов"""
    p1 = TPoly(2, 3)  # 2x^3
    p2 = TPoly(3, 2)  # 3x^2
    p3 = TPoly(1, 3)  # x^3
```

```
    p4 = TPoly(-1, 3) # -x^3

    # 2x^3 + 3x^2
    result1 = p1.add(p2)
    assert result1.coeff(3) == 2
    assert result1.coeff(2) == 3

    # 2x^3 + x^3 = 3x^3
    result2 = p1.add(p3)
    assert result2.coeff(3) == 3
    assert result2.degree == 3

    # 2x^3 + (-x^3) = x^3
    result3 = p1.add(p4)
    assert result3.coeff(3) == 1

    # Сложение с нулевым полиномом
    p_zero = TPoly()
    result4 = p1.add(p_zero)
    assert result4 == p1

def test_multiplication(self):
    """Тест умножения полиномов"""
    # (x + 1) * (x - 1) = x^2 - 1
    p1 = TPoly(1, 1)  # x
    p1 = p1.add(TPoly(1, 0))  # + 1

    p2 = TPoly(1, 1)  # x
    p2 = p2.add(TPoly(-1, 0))  # - 1
```

```
        result = p1.multiply(p2)
        assert result.coeff(2) == 1  # x^2
        assert result.coeff(1) == 0  # x (должен сократиться)
        assert result.coeff(0) == -1  # -1

        # Умножение на нулевой полином
        p_zero = TPoly()
        result2 = p1.multiply(p_zero)
        assert result2.is_zero()

        # Умножение на константу
        p_const = TPoly(5, 0)  # 5
        result3 = p1.multiply(p_const)  # 5*(x+1) = 5x + 5
        assert result3.coeff(1) == 5
        assert result3.coeff(0) == 5

    def test_subtraction(self):
        """Тест вычитания полиномов"""
        p1 = TPoly(2, 3)  # 2x^3
        p2 = TPoly(1, 3)  # x^3

        # 2x^3 - x^3 = x^3
        result = p1.subtract(p2)
        assert result.coeff(3) == 1

        # x^3 - 2x^3 = -x^3
        result2 = p2.subtract(p1)
        assert result2.coeff(3) == -1

        # Вычитание нулевого полинома
```

```
    p_zero = TPoly()
    result3 = p1.subtract(p_zero)
    assert result3 == p1

def test_negate(self):
    """Тест унарного минуса"""
    p1 = TPoly(2, 3)  # 2x^3
    p1 = p1.add(TPoly(-1, 1))  # -x

    neg = p1.negate()  # -2x^3 + x
    assert neg.coeff(3) == -2
    assert neg.coeff(1) == 1
    assert neg.coeff(0) == 0

    # Двойное отрицание должно вернуть исходный полином
    neg2 = neg.negate()
    assert neg2 == p1

def test_equality_poly(self):
    """Тест сравнения полиномов"""
    # x^2 + 2x + 1
    p1 = TPoly(1, 2)  # x^2
    p1 = p1.add(TPoly(2, 1))  # + 2x
    p1 = p1.add(TPoly(1, 0))  # + 1

    # Другой полином с такими же коэффициентами
    p2 = TPoly(1, 2)  # x^2
    p2 = p2.add(TPoly(2, 1))  # + 2x
    p2 = p2.add(TPoly(1, 0))  # + 1
```

```
    # x^2 + 1
    p3 = TPoly(1, 2)  # x^2
    p3 = p3.add(TPoly(1, 0))  # + 1

    assert p1 == p2
    assert not (p1 == p3)
    assert not (p2 == p3)
    assert not (p1 == TPoly())

def test_evaluate_poly(self):
    """Тест вычисления значения полинома"""
    # x^2 + 3x
    p = TPoly(1, 2)  # x^2
    p = p.add(TPoly(3, 1))  # + 3x

    assert abs(p.evaluate(2) - 10) < 0.0001  # 4 + 6 = 10
    assert abs(p.evaluate(0) - 0) < 0.0001
    assert abs(p.evaluate(-1) + 2) < 0.0001  # 1 - 3 = -2

    # Более сложный полином: 2x^3 - x^2 + 4x - 5
    p2 = TPoly(2, 3)    # 2x^3
    p2 = p2.add(TPoly(-1, 2))  # -x^2
    p2 = p2.add(TPoly(4, 1))   # +4x
    p2 = p2.add(TPoly(-5, 0))  # -5

    assert abs(p2.evaluate(1) - 0) < 0.0001   # 2 - 1 + 4 - 5 = 0
    assert abs(p2.evaluate(2) - 15) < 0.0001  # 16 - 4 + 8 - 5 = 15

def test_get_member(self):
    """Тест доступа к членам полинома"""
```

```
    # x^3 + 2x + 1
    p = TPoly(1, 3)  # x^3
    p = p.add(TPoly(2, 1))  # + 2x
    p = p.add(TPoly(1, 0))  # + 1

    # Проверяем члены в порядке убывания степени
    member0 = p.get_member(0)
    assert member0 == (1, 3)  # x^3

    member1 = p.get_member(1)
    assert member1 == (2, 1)  # 2x

    member2 = p.get_member(2)
    assert member2 == (1, 0)  # 1

    # Несуществующий индекс
    assert p.get_member(3) is None
    assert p.get_member(-1) is None

def test_str_representation(self):
    """Тест строкового представления полинома"""
    # Простые полиномы
    assert str(TPoly(1, 2)) == "x^2"
    assert str(TPoly(3, 0)) == "3"
    assert str(TPoly()) == "0"

    # x^2 + 2x + 1
    p1 = TPoly(1, 2)  # x^2
    p1 = p1.add(TPoly(2, 1))  # + 2x
    p1 = p1.add(TPoly(1, 0))  # + 1
```

```
    assert str(p1) == "x^2 + 2x + 1"

    # x^2 - 2x - 1
    p2 = TPoly(1, 2)  # x^2
    p2 = p2.add(TPoly(-2, 1))  # - 2x
    p2 = p2.add(TPoly(-1, 0))  # - 1
    assert str(p2) == "x^2 - 2x - 1"

    # 2x^3 - x^2
    p3 = TPoly(2, 3)    # 2x^3
    p3 = p3.add(TPoly(-1, 2))  # - x^2
    assert str(p3) == "2x^3 - x^2"

def test_operator_overloading(self):
    """Тест перегрузки операторов"""
    p1 = TPoly(1, 2)  # x^2
    p2 = TPoly(2, 1)  # 2x

    # Сложение
    result_add = p1 + p2
    assert result_add.coeff(2) == 1
    assert result_add.coeff(1) == 2

    # Вычитание
    result_sub = p1 - p2
    assert result_sub.coeff(2) == 1
    assert result_sub.coeff(1) == -2

    # Умножение
    result_mul = p1 * p2  # x^2 * 2x = 2x^3
```

```
    assert result_mul.coeff(3) == 2

    # Унарный минус
    result_neg = -p1  # -x^2
    assert result_neg.coeff(2) == -1

def test_normalization(self):
    """Тест нормализации полинома"""
    # Создаем ненормализованный полином: x^2 + 0x + x^2 + (-x^2) + 3
    # Вместо цепочки add, создадим полином напрямую через члены
    poly = TPoly()

    # Добавляем члены через внутренний список (для тестирования)
    poly._members.append(TMember(1, 2))   # x^2
    poly._members.append(TMember(0, 1))   # 0x (должен удалиться)
    poly._members.append(TMember(1, 2))   # + x^2 (должен сложиться с первым)
    poly._members.append(TMember(-1, 2))  # + (-x^2) (должен сократиться)
    poly._members.append(TMember(3, 0))   # + 3

    # Вызываем нормализацию
    poly._normalize()

    # После нормализации должен остаться только 3
    # Все x^2 должны сложиться: 1 + 1 - 1 = 1, но в тесте ожидаем 0
    # Перепишем тест: x^2 + x^2 - x^2 = x^2, плюс константа 3
    assert poly.coeff(2) == 1  # x^2
    assert poly.coeff(1) == 0
    assert poly.coeff(0) == 3
    assert len(poly._members) == 2  # x^2 и 3
```

```
def test_complex_polynomial():
    """Тест сложного полинома и операций с ним"""
    # Создаем полином: 3x^4 - 2x^3 + 5x^2 - 7x + 4
    p = TPoly(3, 4)
    p = p.add(TPoly(-2, 3))
    p = p.add(TPoly(5, 2))
    p = p.add(TPoly(-7, 1))
    p = p.add(TPoly(4, 0))

    # Проверяем коэффициенты
    assert p.coeff(4) == 3
    assert p.coeff(3) == -2
    assert p.coeff(2) == 5
    assert p.coeff(1) == -7
    assert p.coeff(0) == 4

    # Производная: 12x^3 - 6x^2 + 10x - 7
    deriv = p.differentiate()
    print(f"Производная: {deriv}")
    print(f"coeff(3): {deriv.coeff(3)} (ожидается 12)")
    print(f"coeff(2): {deriv.coeff(2)} (ожидается -6)")
    print(f"coeff(1): {deriv.coeff(1)} (ожидается 10)")
    print(f"coeff(0): {deriv.coeff(0)} (ожидается -7)")

    assert deriv.coeff(3) == 12
    assert deriv.coeff(2) == -6
    assert deriv.coeff(1) == 10
    assert deriv.coeff(0) == -7
```

```
# Вычисление в точке
assert abs(p.evaluate(1) - 3) < 0.0001  # 3-2+5-7+4 = 3
assert abs(p.evaluate(0) - 4) < 0.0001

# Умножение на себя (квадрат полинома)
square = p.multiply(p)
# Проверяем старшую степень: (3x^4)*(3x^4) = 9x^8
assert square.degree == 8
assert square.coeff(8) == 9
```

3. Тестовые наборы данных для тестирования класса

1. Конструктор (test_constructor)

Описание теста	Входные данные	Ожидаемые результаты
Одночлен с положительными коэффициентом и степенью	coeff=6, degree=3	coeff=6, degree=3
Константа (нулевая степень)	coeff=3, degree=0	coeff=3, degree=0
Нуль-полином по умолчанию	-	coeff=0, degree=0
Нуль-полином при нулевом коэффициенте	coeff=0, degree=5	coeff=0, degree=0

2. Сравнение (test_equality)

Тестовый случай	Одночлен 1	Одночлен 2	Ожидаемый результат
Полное совпадение	TMember(5, 3)	TMember(5, 3)	True
Разная степень	TMember(5, 3)	TMember(5, 2)	False
Разный коэффициент	TMember(5, 3)	TMember(4, 3)	False
Оба разные	TMember(5, 3)	TMember(4, 2)	False

3. Дифференцирование (test_differentiate)

Исходный одночлен	Производная	Обоснование
x^3 (1, 3)	3x^2 (3, 2)	d/dx(x^3) = 3x^2
5x^2 (5, 2)	10x (10, 1)	d/dx(5x^2) = 10x
Константа 7 (7, 0)	0 (0, 0)	d/dx(const) = 0
2x (2, 1)	2 (2, 0)	d/dx(2x) = 2

4. Вычисление значения (test_evaluate)

Одночлен	x	Ожидаемый результат	Обоснование
3x^2	2	12	3*4 = 12
3x^2	0	0	3*0 = 0
3x^2	-1	3	3*1 = 3
Константа 5	10	5	const
Константа 5	-5	5	const
-2x^3	3	-54	-2*27 = -54

5. Строковое представление (test_to_string)

Одночлен	Строковое представление
(1, 3)	"x^3"
(-1, 3)	"-x^3"
(5, 3)	"5x^3"
(-5, 3)	"-5x^3"
(1, 1)	"x"
(-1, 1)	"-x"
(7, 0)	"7"
(-7, 0)	"-7"
(0, 5)	"0"
(1, 0)	"1"

Одночлен	Строковое представление
(-1, 0)	"-1"

6. Сеттеры (test_setters)

Действие	Начальное состояние	Изменение	Конечное состояние
Изменение коэффициента	(2, 3)	coeff=5	(5, 3)
Изменение степени	(5, 3)	degree=4	(5, 4)
Установка нулевого коэффициента	(5, 4)	coeff=0	(0, 0)

Тесты для класса TPoly

1. Конструктор (test_constructor)

Описание	Входные данные	Ожидаемый degree	Ожидаемый coeff(n)
Одночленный полином	(6, 3)	3	coeff(3)=6
Константа	(3, 0)	0	coeff(0)=3
Нуль-полином по умолчанию	-	0	is_zero()=True
Нуль-полином при нулевом коэффициенте	(0, 5)	0	is_zero()=True

2. Степень полинома (test_degree)

Полином	Ожидаемая степень	Комментарий
x^2 + 1	2	Наибольшая степень 2
Константа 17	0	Нет переменных
Нуль-полином	0	По определению

3. Получение коэффициентов (test_coeff)

Полином: x^3 + 2x + 1	Степень n	Ожидаемый coeff(n)
x^3 + 2x + 1	4	0
x^3 + 2x + 1	3	1
x^3 + 2x + 1	2	0
x^3 + 2x + 1	1	2
x^3 + 2x + 1	0	1

4. Сложение (test_addition)

Случай	Полином 1	Полином 2	Результат
Разные степени	2x^3	3x^2	2x^3 + 3x^2
Одинаковые степени	2x^3	x^3	3x^3
Взаимное уничтожение	2x^3	-x^3	x^3
С нулевым полиномом	2x^3	0	2x^3

5. Умножение (test_multiplication)

Случай	Полином 1	Полином 2	Результат	Проверяемые коэффициенты
(x+1)*(x-1)	x + 1	x - 1	x^2 - 1	coeff(2)=1, coeff(1)=0, coeff(0)=-1
На нулевой полином	x + 1	0	0	is_zero()=True
На константу	x + 1	5	5x + 5	coeff(1)=5, coeff(0)=5

6. Вычитание (test_subtraction)

Случай	Полином 1	Полином 2	Результат	coeff(3)
2x^3 - x^3	2x^3	x^3	x^3	1
x^3 - 2x^3	x^3	2x^3	-x^3	-1
Минус ноль	2x^3	0	2x^3	2

7. Унарный минус (test_negate)

Исходный полином	Результат -p	Проверяемые коэффициенты
2x^3 - x	-2x^3 + x	coeff(3) = -2, coeff(1) = 1

8. Вычисление значения (test_evaluate_poly)

Полином	x	Ожидаемый результат	Обоснование
x^2 + 3x	2	10	4 + 6 = 10
x^2 + 3x	0	0	0 + 0 = 0
x^2 + 3x	-1	-2	1 - 3 = -2

Полином	x	Ожидаемый результат	Обоснование
2x^3 - x^2 + 4x - 5	1	0	2 - 1 + 4 - 5 = 0
2x^3 - x^2 + 4x - 5	2	15	16 - 4 + 8 - 5 = 15

9. Доступ к членам (test_get_member)

Полином: x^3 + 2x + 1	Индекс	Ожидаемый результат
x^3 + 2x + 1	0	(1, 3)
x^3 + 2x + 1	1	(2, 1)
x^3 + 2x + 1	2	(1, 0)
x^3 + 2x + 1	3	None
x^3 + 2x + 1	-1	None

10. Строковое представление (test_str_representation)

Полином	Строковое представление
x^2	"x^2"
3	"3"
0	"0"
x^2 + 2x + 1	"x^2 + 2x + 1"
x^2 - 2x - 1	"x^2 - 2x - 1"
2x^3 - x^2	"2x^3 - x^2"

11. Операторы (test_operator_overloading)

Операция	Полином 1	Полином 2	Проверяемые коэффициенты
p1 + p2	x^2	2x	coeff(2)=1, coeff(1)=2
p1 - p2	x^2	2x	coeff(2)=1, coeff(1)=-2
p1 * p2	x^2	2x	coeff(3)=2
-p1	x^2	-	coeff(2)=-1

12. Нормализация (test_normalization)

Исходные члены	Ожидаемый результат после нормализации
x^2 + 0x + x^2 + (-x^2) + 3	x^2 + 3 (coeff(2)=1, coeff(0)=3)

13. Комплексный полином (test_complex_polynomial)

Операция	Полином: 3x^4 - 2x^3 + 5x^2 - 7x + 4	Ожидаемый результат
Коэффициенты	-	coeff(4)=3, coeff(3)=-2, coeff(2)=5, coeff(1)=-7, coeff(0)=4
Производная	-	12x^3 - 6x^2 + 10x - 7
Вычисление в x=1	-	3
Вычисление в x=0	-	4
Квадрат полинома	p * p	Степень 8, coeff(8)=9

14. Производная полинома

Полином	Производная	Проверяемые точки
3x^4 - 2x^3 + 5x^2 - 7x + 4	12x^3 - 6x^2 + 10x - 7	coeff(3)=12, coeff(2)=-6, coeff(1)=10, coeff(0)=-7