

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
Современные технологии программирования

Практическая работа №9
«Абстрактный тип данных (ADT) «Полином»»

Выполнил: студент 4 курса
группы ИП-111
Кузьменок Денис Витальевич

Проверил преподаватель:
Зайцев Михаил Георгиевич

Новосибирск, 2024 г.

Цель

Сформировать практические навыки реализации абстрактных типов данных с помощью классов и шаблонов классов STL.

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 TPolу

Данные

Полиномы TPolу — это неизменяемые полиномы с целыми коэффициентами.

Операции

Операции могут вызываться только объектом «полином» (тип TPolу), указатель на который передаётся в них по умолчанию. При описании операций этот объект в разделе «Вход» не указывается.

Таблица 1. Описание операций на ADT TPolу

Наименование операции	Описание
Конструктор	
Начальные значения:	Коэффициент (с) и степень (n) одночленного полинома
Процесс:	Создаёт одночленный полином с коэффициентом (с) и степенью (n), или ноль-полином, если коэффициент (с) 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 (с). Степень нулевого полинома равна 0. Например: $a = (x^2+1)$, а.Степень = 2 $a = (17)$, а. Степень = 0
Выход:	n - целое число - степень полинома.
Постусловия:	Нет.
Коэффициент	
Вход:	n - целое число - степень полинома.
Предусловия:	Полином – не нулевой.
Процесс:	Отыскивает коэффициент (с) при члене полинома со степенью n ($c \cdot x^n$). Возвращает коэффициент (с) 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.

Спецификация абстрактного типа данных «Одночлен».

ADT TMember

Данные

Одночлен TMember - это изменяемые одночленные полиномы с целыми коэффициентами. Коэффициент и степень хранятся в полях целого типа FCoeff и FDegree соответственно.

Операции

Операции могут вызываться только объектом «одночлен» (тип TMember), указатель на который передаётся в них по умолчанию. При описании операций этот объект в разделе «Вход» не указывается.

Таблица 2. Описание операций на ADT TMember.

Наименование операции	Описание
Конструктор	
Начальные значения:	Коэффициент (с) и степень (n) одночленного полинома
Процесс:	Создаёт одночленный полином с коэффициентом (с) и степенью (n), или ноль-полином, если коэффициент (с) равен 0 и возвращает указатель на него. Например: Конструктор(6,3 = $6x^3$ Конструктор(3,0) = 3 Конструктор() = 0
Выход:	Указатель на созданный одночленный полином.
Постусловия:	Нет.
ЧитатьСтепень	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает степень n одночленного полинома (содержимое поля FDegree). Степень нулевого полинома равна 0. Например: $a = (1x^2)$, a.Степень = 2
Выход:	n - целое число - степень полинома.
Постусловия:	Нет.

ПисатьСтепень	
Вход:	n - целое число - степень полинома.
Предусловия:	Нет.
Процесс:	Записывает степень n одночленного полинома в поле FDegree.
Выход:	Нет.
Постусловия:	Поле FDegree = n.
ПисатьКоэффициент	
Вход:	c - целое число - коэффициент полинома.
Предусловия:	Нет.
Процесс:	Записывает коэффициент c одночленного полинома в поле FCoeff.
Выход:	Нет.
Постусловия:	Поле FCoeff = c.
Равно	
Вход:	q - одночлен.
Предусловия:	Нет.
Процесс:	Сравнивает одночлен с одночленом q на равенство. Возвращает значение True, если одночлены равны, т.е. имеют одинаковые коэффициенты и степени, и значение False - в противном случае.
Выход:	Булево значение.
Постусловия:	Нет.
Дифференцировать	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт одночлен, являющийся производной одночлена и возвращает его. Например: $a = (x^3)$, а.Дифференцировать = $3x^2$.
Выход:	Одночлен.
Постусловия:	Нет.

Вычислить	
Вход:	x – действительное число.
Предусловия:	Нет.
Процесс:	Вычисляет значение одночлена в точке x и возвращает его. Например: $a = (1x^2)$, а.Вычислить(2) = 4.
Выход:	Действительное число.
Постусловия:	Нет.
ОдночленВСтроку	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Формирует строковое представление одночлена.
Выход:	Строка.
Постусловия:	Нет.

Реализация:

```
Первый полином = 11x^51-5x^7+8x^6
Второй полином = 20x^130-6x^6

Первый полином с обратным знаком = -11x^51+5x^7-8x^6
Второй полином с добавлением элемента той же степени = 20x^130-3x^6
Произведение первого полинома на второй = 220x^181-100x^137+160x^136-33x^57+15x^13-24x^12
Разница первого полинома и второго = -20x^130+11x^51-5x^7+11x^6
Равны ли эти полиндромы между собой?
Нет!
Производная от первого полинома = 561x^50-35x^6+48x^5
Производная от второго полинома = 2600x^129-18x^5

Результат первого полинома в точке 6.31 = 6,96450102451812E+41
Результат второго полинома в точке -5.4 = 3,25251098973086E+96

Элемент по индексу 2 из первого полинома = (8, 6)
Элемент по индексу 1 из второго полинома = (-3, 6)

Очистили полином, содержащий в себе произведение первого на второй:

Наибольшая степень в первом полиноме = 51
Наибольшая степень во втором полиноме = 130

Коэффициент стоящий у наибольшей степени в первом полиноме = 11
Коэффициент стоящий у наибольшей степени во втором полиноме = 20
```

Рис. 1 – Результат проверки работоспособности программы.

Обозреватель тестов

Выполнение тестов: завершено 18 из 19 тестов

Тестирование	Длительность	Признаки	Сообщение об ошибке
PolynomTests (19)	99 мс		
PolynomTests (1...	99 мс		
PolynomTests (..	99 мс		
TestAdd1	< 1 мс		
TestAdd2	< 1 мс		
TestAdd3	< 1 мс		
TestCalc1	1 мс		
TestCalc2	< 1 мс		
TestClear	< 1 мс		
TestDiff1	< 1 мс		
TestElement...	< 1 мс		
TestElement...	22 мс		
TestEquals1	< 1 мс		
TestEquals2	1 мс		
TestMinus	< 1 мс		
TestMul1	3 мс		
TestMul2	< 1 мс		
TestRetCoeff1	72 мс		
TestRetCoeff2	< 1 мс		
TestRetDegr...	< 1 мс		
TestSub1	< 1 мс		
TestSub2	< 1 мс		

Выполнить

Подробная сводка по

TestRetCoeff1

Источник: У

Длительность

Рис. 2 – Результат выполнения модульных тестов.

1. TestAdd1

- **Что проверяет:** создание объекта типа `TPoly` и добавление начальных значений.
- **Входные значения:**
 - `TPoly tpoly = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly.Members.Add(new TMember(1, 0));` - задание начальных значений.
- **Ожидаемое значение:** " $1x^0$ " - полином.

2. TestAdd2

- **Что проверяет:** создание объекта типа `TPoly` и добавление нескольких значений.
- **Входные значения:**
 - `TPoly tpoly = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly.Members.Add(new TMember(1, 0));`
 - `tpoly.Members.Add(new TMember(1, 1));`
- **Ожидаемое значение:** " $1x^1 + 1x^0$ " - полином.

3. TestAdd3

- **Что проверяет:** сложение чисел в полиноме с одинаковыми степенями.
- **Входные значения:**
 - `TPoly tpoly = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly.Members.Add(new TMember(2, 5));`
 - `tpoly.Members.Add(new TMember(5, 5));`
- **Ожидаемое значение:** " $7x^5$ " - полином.

4. TestMul1

- **Что проверяет:** Произведение двух полиномов.

- **Входные значения:**

- `TPoly tpoly = new TProc()` — создаётся объект класса `TPoly`.
- `TPoly newtpoly = new TPoly()` – создание второго объекта класса `TPoly`.
- `tpoly.Members.Add(new TMember(1, 0));`
- `tpoly.Members.Add(new TMember(1, 2));`
- `newtpoly.Members.Add(new TMember(2, 0));`
- `newtpoly.Members.Add(new TMember(1, 1));`

- **Ожидаемое значение:** “ $1x^3+2x^2+1x^1+2x^0$ ” - полином.

5. TestMul2

- **Что проверяет:** Произведение двух полиномов.

- **Входные значения:**

- `TPoly tpoly = new TProc()` — создаётся объект класса `TPoly`.
- `TPoly newtpoly = new TPoly()` – создание второго объекта класса `TPoly`.
- `tpoly.Members.Add(new TMember(1, 0));`
- `tpoly.Members.Add(new TMember(1, 1));`
- `newtpoly.Members.Add(new TMember(1, 0));`
- `newtpoly.Members.Add(new TMember(1, 1));`

- **Ожидаемое значение:** “ $1x^2+2x^1+1x^0$ ” - полином.

6. TestSub1

- **Что проверяет:** Вычитание двух полиномов.

- **Входные значения:**

- `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
- `TPoly tpoly2 = new TPoly()` – создание второго объекта класса `TPoly`.
- `tpoly.Members.Add(new TMember(-21, 4));`
- `tpoly.Members.Add(new TMember(6, 9));`
- `newtpoly.Members.Add(new TMember(8, 7));`
- `newtpoly.Members.Add(new TMember(-12, 43));`
- **Ожидаемое значение:** “ $12x^{43}+6x^9-8x^7-21x^4$ ” - полином.

7. TestSub2

- **Что проверяет:** Вычитание двух полиномов с одинаковыми степенями.
- **Входные значения:**
 - `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
 - `TPoly tpoly2 = new TPoly()` – создание второго объекта класса `TPoly`.
 - `tpoly.Members.Add(new TMember(-21, 4));`
 - `tpoly.Members.Add(new TMember(6, 4));`
 - `newtpoly.Members.Add(new TMember(8, 4));`
 - `newtpoly.Members.Add(new TMember(-12, 4));`
- **Ожидаемое значения:** “ $-11x^4$ ” – полином.

8. TestClear

- **Что проверяет:** Очищение полинома от значений.
- **Входные значения:**
 - `TPoly tpoly = new TProc()` — создаётся объект класса `TPoly`.

- `tpoly.Members.Add(new TMember(1, 0));`
- `tpoly.Clear()`

- **Ожидаемое значения:** пустой `SortedSet<TMember>`.

9. TestCalc1

- **Что проверяет:** вычисление значения полинома в заданной точке.
- **Входные значения:**
 - `TPoly tpoly = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly.Members.Add(new TMember(1, 2));`
 - `tpoly.Members.Add(new TMember(3, 3));`
 - `tpoly.Members.Add(new TMember(4, 2));`
- **Ожидаемое значения:** 44 — результат вычисления.

10. TestCalc2

- **Что проверяет:** вычисление значения полинома в заданной точке.
- **Входные значения:**
 - `TPoly tpoly = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly.Members.Add(new TMember(1, 2));`
 - `tpoly.Members.Add(new TMember(3, 0));`
 - `tpoly.Members.Add(new TMember(4, 0));`
- **Ожидаемое значения:** 11 — результат вычисления.

11. TestEquals1

- **Что проверяет:** равны ли два полинома между собой.
- **Входные значения:**
 - `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
 - `TPoly tpoly2 = new TPoly()` — создание второго объекта класса `TPoly`.

- `tpoly1.Members.Add(new TMember(1, 2));`
 - `tpoly2.Members.Add(new TMember(1, 2));`
 - **Ожидаемое значения:** `true` — два полинома равны между собой.
- 12. TestEquals2**
- **Что проверяет:** равны ли два полинома между собой.
 - **Входные значения:**
 - `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
 - `TPoly tpoly2 = new TPoly()` – создание второго объекта класса `TPoly`.
 - `tpoly1.Members.Add(new TMember(1, 2));`
 - `tpoly2.Members.Add(new TMember(0, 2));`
 - **Ожидаемое значения:** `false` — два полинома не равны между собой.
- 13. TestDiff1**
- **Что проверяет:** вычисляет дифференциал от полинома.
 - **Входные значения:**
 - `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly1.Members.Add(new TMember(1, 3));`
 - **Ожидаемое значения:** `"3x^2"` — результат нахождения производной от полинома.
- 14. TestElementAt1**
- **Что проверяет:** взятие элемента по необходимому индексу.
 - **Входные значения:**
 - `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly1.Members.Add(new TMember(11, 11));`
 - `tpoly1.Members.Add(new TMember(22, 22));`

- **Ожидаемое значения:** (11, 11) — пара чисел, находящихся на позиции 1 в полиноме.
- 15. TestElementAt2**
- **Что проверяет:** взятие элемента по необходимому индексу.
 - **Входные значения:**
 - `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly1.Members.Add(new TMember(-21, 33));`
 - `tpoly1.Members.Add(new TMember(6, -4));`
 - `tpoly1.Members.Add(new TMember(10, 6));`
 - **Ожидаемое значения:** генерация исключения `ArgumentOutOfRangeException` — т.к. невозможно взять элемент по индексу 5.
- 16. TestRetDegree**
- **Что проверяет:** возвращает наибольшую степень в полиноме.
 - **Входные значения:**
 - `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly1.Members.Add(new TMember(11, 11));`
 - `tpoly1.Members.Add(new TMember(22, 22));`
 - **Ожидаемое значения:** 22 — наибольшая степень в полиноме.
- 17. TestRetCoeff1**
- **Что проверяет:** возвращает коэффициент у переданной степени члена в полиноме.
 - **Входные значения:**
 - `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly1.Members.Add(new TMember(3, 7));`
 - `tpoly1.Members.Add(new TMember(-84, 15));`

- **Ожидаемое значения:** -84 — т.к. у переданной степени 15 коэффициент перед x равен -84.

18. TestRetCoeff2

- **Что проверяет:** возвращает коэффициент у переданной степени члена в полиноме.
- **Входные значения:**
 - `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly1.Members.Add(new TMember(-7, 32));`
 - `tpoly1.Members.Add(new TMember(7, 6));`
 - `tpoly1.Members.Add(new TMember(112, 2));`
- **Ожидаемое значения:** генерация исключения `InvalidOperationException` — т.к. нет коэффициента перед степенью 33.
-

19. TestMinus

- **Что проверяет:** заменяет знаки в полиноме на противоположные.
- **Входные значения:**
 - `TPoly tpoly1 = new TProc()` — создаётся объект класса `TPoly`.
 - `tpoly1.Members.Add(new TMember(11, 1));`
 - `tpoly1.Members.Add(new TMember(-22, 3));`
 - `tpoly1.Members.Add(new TMember(20, 2));`
- **Ожидаемое значения:** “ $22x^3 - 20x^2 - 11x^1$ ” — результат изменения знака на противоположный.

Вывод

В результате работы над лабораторной работой были сформированы практические навыки реализации параметризованного абстрактного типа данных на языке C#, разработки функций классов на языке C#, разработка модульных тестов для тестирования функций классов и выполнения модульного тестирования на языке C# с помощью средств автоматизации Visual Studio.

Листинг программы:

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab9
{
    class Program
    {
        static void Main(string[] args)
        {
            TPoly tPoly1 = new TPoly();
            TPoly tPoly2 = new TPoly();
            tPoly1.Members.Add(new TMember(-5, 7));
            tPoly1.Members.Add(new TMember(8, 6));
            tPoly1.Members.Add(new TMember(11, 51));
            tPoly2.Members.Add(new TMember(-6, 6));
            tPoly2.Members.Add(new TMember(20, 130));

            string firstPolynome = tPoly1.Show();
            string secondPolynome = tPoly2.Show();
            string resultMinus = tPoly1.Minus().Show();
            Console.WriteLine($"Первый полином = {firstPolynome}\nВторой полином = {secondPolynome}\n");
            Console.WriteLine($"Первый полином с обратным знаком = {resultMinus}");

            tPoly2.Members.Add(new TMember(3, 6));
            string secondPolynomeAdd = tPoly2.Show();
            Console.WriteLine($"Второй полином с добавлением элемента той же степени = {secondPolynomeAdd}");

            TPoly mul = tPoly1.Mul(tPoly2);
            string mulPolynomes = mul.Show();
            Console.WriteLine($"Произведение первого полинома на второй = {mulPolynomes}");

            TPoly sub = tPoly1.Sub(tPoly2);
            string subPolynomes = sub.Show();
            Console.WriteLine($"Разница первого полинома и второго = {subPolynomes}");

            Console.WriteLine($"Равны ли эти полиндромы между собой? {(tPoly1.Equals(tPoly2) ? "\nДа!" : "\nНет!")}");

            string diffFirstPolynome = tPoly1.Diff().Show();
            string diffSecondPolynome = tPoly2.Diff().Show();
            Console.WriteLine($"Производная от первого полинома = {diffFirstPolynome}\nПроизводная от второго полинома = {diffSecondPolynome}\n");

            double result1 = tPoly1.Calculate(6.31);
            double result2 = tPoly2.Calculate(-5.4);
            Console.WriteLine($"Результат первого полинома в точке 6.31 = {result1}\nРезультат второго полинома в точке -5.4 = {result2}\n");
        }
    }
}
```

```

        Tuple<int, int> fromFirstPolynome = tPoly1.TakeElement(2);
        Tuple<int, int> fromSecondPolynome = tPoly2.TakeElement(1);
        Console.WriteLine($"Элемент по индексу 2 из первого полинома =
{fromFirstPolynome}\nЭлемент по индексу 1 из второго полинома =
{fromSecondPolynome}\n");

        sub.Clear();
        string subClear = sub.Show();
        Console.WriteLine($"Очистили полином, содержащий в себе
произведение первого на второй: {subClear}\n");

        int degreeFirst = tPoly1.ReturnDegree();
        int degreeSecond = tPoly2.ReturnDegree();
        Console.WriteLine($"Наибольшая степень в первом полиноме =
{degreeFirst}\nНаибольшая степень во втором полиноме = {degreeSecond}\n");

        int maxCoeffFirst = tPoly1.ReturnCoefficient(degreeFirst);
        int maxCoeffSecond = tPoly2.ReturnCoefficient(degreeSecond);
        Console.WriteLine($"Коэффициент стоящий у наибольшей степени в
первом полиноме = {maxCoeffFirst}\nКоэффициент стоящий у наибольшей степени во
втором полиноме = {maxCoeffSecond}\n");
    }
}

```

TMember.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab9
{
    public class TMember : IComparable<TMember>
    {
        private int _fcoefficient;
        private int _fdegree;

        public int FCoefficient
        {
            get { return _fcoefficient; }
            set
            {
                if (value == 0)
                {
                    _fdegree = 0;
                }
                _fcoefficient = value;
            }
        }

        public int FDegree
        {
            get { return _fdegree; }
            set
            {
                if (_fcoefficient == 0)
                {
                    _fdegree = 0;
                }
                else _fdegree = value;
            }
        }

        public TMember(int coeff = 0, int degree = 0)
        {
            FCoefficient = coeff;
            FDegree = degree;
        }

        public override bool Equals(object obj)
        {
            if (((TMember)obj).FCoefficient == this.FCoefficient) &&
                (((TMember)obj).FDegree == this.FDegree))
                return true;
            else
                return false;
        }

        public TMember Differentiate()
        {
            return new TMember()
            {

```

```

        FCoefficient = (this.FCoefficient * this.FDegree),
        FDegree = (this.FDegree - 1)
    };
}

public double Calculate(double a)
{
    return this.FCoefficient * Math.Pow(a, this.FDegree);
}

public string TMemberToString()
{
    return (this.FCoefficient == 0) ? "" :
    $"{this.FCoefficient}x^{this.FDegree}";
}

public int CompareTo(TMember other)
{
    if (this.FDegree.CompareTo(other.FDegree) != 0)
        return this.FDegree.CompareTo(other.FDegree);
    else
    {
        other.FCoefficient += this.FCoefficient;
        return 0;
    }
}
}
}

```

TPoly.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab9
{
    public class TPoly
    {
        public SortedSet<TMember> Members;

        public TPoly()
        {
            Members = new SortedSet<TMember>();
            Members.Add(new TMember(0, 0));
        }

        public TPoly(int c, int n)
        {
            Members = new SortedSet<TMember>();
            Members.Add(new TMember(c, n));
        }

        public TPoly Add(TPoly other)
        {
            TPoly newPoly = new TPoly();

            foreach(TMember m in other.Members)
            {
                newPoly.Members.Add(new TMember(m.FCoefficient, m.FDegree));
            }

            foreach(TMember m in this.Members)
            {
                newPoly.Members.Add(new TMember(m.FCoefficient, m.FDegree));
            }

            return newPoly;
        }

        public TPoly Mul(TPoly other)
        {
            TPoly newPoly = new TPoly();

            foreach(TMember m in other.Members)
            {
                foreach(TMember newM in this.Members)
                {
                    newPoly.Members.Add(new TMember(newM.FCoefficient *
m.FCoefficient, newM.FDegree + m.FDegree));
                }
            }

            return newPoly;
        }
    }
}
```

```

public TPolynomial Sub(TPolynomial other)
{
    TPolynomial newPoly = new TPolynomial();

    foreach(TMember m in other.Members)
    {
        newPoly.Members.Add(new TMember(-m.FCoefficient, m.FDegree));
    }

    foreach(TMember m in this.Members)
    {
        newPoly.Members.Add(new TMember(m.FCoefficient, m.FDegree));
    }

    return newPoly;
}

public TPolynomial Minus()
{
    TPolynomial newPoly = new TPolynomial();

    foreach(TMember m in this.Members)
    {
        newPoly.Members.Add(new TMember(-m.FCoefficient, m.FDegree));
    }

    return newPoly;
}

public override bool Equals(object obj)
{
    if (((TPolynomial) obj).Members.SequenceEqual(this.Members))
        return true;
    else
        return false;
}

public TPolynomial Diff()
{
    TPolynomial newPoly = new TPolynomial();

    foreach(TMember m in this.Members)
    {
        newPoly.Members.Add(new TMember(m.FCoefficient,
m.FDegree).Differentiate());
    }

    return newPoly;
}

public double Calculate(double a)
{
    double result = 0.0;

    foreach(TMember m in this.Members)
    {
        result += m.Calculate(a);
    }
}

```

```

        return result;
    }

    public Tuple<int, int> TakeElement(int i)
    {
        if(i > 0 && i < Members.Count)
        {
            return Tuple.Create(
                this.Members.Reverse().ElementAt(i).FCoefficient,
                this.Members.Reverse().ElementAt(i).FDegree);
        }
        else
        {
            throw new ArgumentOutOfRangeException();
        }
    }

    public void Clear()
    {
        Members = new SortedSet<TMember>
        {
            new TMember(0, 0),
        };
    }

    public int ReturnDegree()
    {
        return Members.Last().FDegree;
    }

    public int ReturnCoefficient(int n)
    {
        if(n >= Members.First().FDegree && n <= Members.Last().FDegree)
        {
            return Members.Single(x => x.FDegree == n).FCoefficient;
        }
        else
        {
            throw new InvalidOperationException();
        }
    }

    public string Show()
    {
        string str = "";

        foreach(TMember m in this.Members.Reverse())
        {
            str += (m.FCoefficient > 0) ? "+" : "";
            str += m.TMemberToString();
        }

        return str.TrimStart('+');
    }
}

```


UnitTests1.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using lab9;

namespace PolynomTests
{
    [TestClass]
    public class PolynomTests
    {
        [TestMethod]
        public void TestAdd1()
        {
            TPoly tpoly = new TPoly();
            tpoly.Members.Add(new TMember(1, 0));

            string actual = "1x^0";

            Assert.AreEqual(tpoly.Show(), actual);
        }

        [TestMethod]
        public void TestAdd2()
        {
            TPoly tpoly = new TPoly();
            tpoly.Members.Add(new TMember(1, 0));
            tpoly.Members.Add(new TMember(1, 1));

            string actual = "1x^1+1x^0";

            Assert.AreEqual(tpoly.Show(), actual);
        }

        [TestMethod]
        public void TestAdd3()
        {
            TPoly tpoly1 = new TPoly();
            tpoly1.Members.Add(new TMember(2, 5));
            tpoly1.Members.Add(new TMember(5, 5));

            string actual = "7x^5";

            Assert.AreEqual(tpoly1.Show(), actual);
        }

        [TestMethod]
        public void TestMul1()
        {
            TPoly tpoly = new TPoly();
            TPoly newtpoly = new TPoly();
            tpoly.Members.Add(new TMember(1, 0));
            tpoly.Members.Add(new TMember(1, 1));
            newtpoly.Members.Add(new TMember(1, 0));
            newtpoly.Members.Add(new TMember(1, 1));

            TPoly addpoly = tpoly.Add(newtpoly);
            addpoly = tpoly.Mul(newtpoly);

            string actual = "1x^2+2x^1+1x^0";
        }
    }
}
```

```

        Assert.AreEqual(addpoly.Show(), actual);
    }

    [TestMethod]
    public void TestMul2()
    {
        TPoly tpoly = new TPoly();
        TPoly newtpoly = new TPoly();
        tpoly.Members.Add(new TMember(1, 0));
        tpoly.Members.Add(new TMember(1, 2));
        newtpoly.Members.Add(new TMember(2, 0));
        newtpoly.Members.Add(new TMember(1, 1));

        TPoly addpoly = tpoly.Add(newtpoly);
        addpoly = tpoly.Mul(newtpoly);

        string actual = "1x^3+2x^2+1x^1+2x^0";

        Assert.AreEqual(addpoly.Show(), actual);
    }

    [TestMethod]
    public void TestSub1()
    {
        TPoly tpoly1 = new TPoly();
        TPoly tpoly2 = new TPoly();
        tpoly1.Members.Add(new TMember(-21, 4));
        tpoly1.Members.Add(new TMember(6, 9));
        tpoly2.Members.Add(new TMember(8, 7));
        tpoly2.Members.Add(new TMember(-12, 43));

        TPoly subpoly = tpoly1.Sub(tpoly2);

        string actual = "12x^43+6x^9-8x^7-21x^4";

        Assert.AreEqual(subpoly.Show(), actual);
    }

    [TestMethod]
    public void TestSub2()
    {
        TPoly tpoly1 = new TPoly();
        TPoly tpoly2 = new TPoly();
        tpoly1.Members.Add(new TMember(-21, 4));
        tpoly1.Members.Add(new TMember(6, 4));
        tpoly2.Members.Add(new TMember(8, 4));
        tpoly2.Members.Add(new TMember(-12, 4));

        TPoly subpoly = tpoly1.Sub(tpoly2);

        string actual = "-11x^4";

        Assert.AreEqual(subpoly.Show(), actual);
    }

    [TestMethod]
    public void TestClear()
    {

```

```

TPoly tpoly = new TPoly();
tpoly.Members.Add(new TMember(1, 0));
tpoly.Clear();

string actual = "";

Assert.AreEqual(tpoly.Show(), actual);
}

[TestMethod]
public void TestCalc1()
{
    TPoly tpoly = new TPoly();
    tpoly.Members.Add(new TMember(1, 2));
    tpoly.Members.Add(new TMember(3, 3));
    tpoly.Members.Add(new TMember(4, 2));

    double actual = 44;

    Assert.AreEqual(tpoly.Calculate(2), actual);
}

[TestMethod]
public void TestCalc2()
{
    TPoly tpoly = new TPoly();
    tpoly.Members.Add(new TMember(1, 2));
    tpoly.Members.Add(new TMember(3, 0));
    tpoly.Members.Add(new TMember(4, 0));

    double actual = 11;

    Assert.AreEqual(tpoly.Calculate(2), actual);
}

[TestMethod]
public void TestEquals1()
{
    TPoly tpoly1 = new TPoly();
    tpoly1.Members.Add(new TMember(1, 2));

    TPoly tpoly2 = new TPoly();
    tpoly2.Members.Add(new TMember(1, 2));

    Assert.IsTrue(tpoly1.Equals(tpoly2));
}

[TestMethod]
public void TestEquals2()
{
    TPoly tpoly1 = new TPoly();
    tpoly1.Members.Add(new TMember(1, 2));

    TPoly tpoly2 = new TPoly();
    tpoly2.Members.Add(new TMember(0, 2));

    Assert.IsFalse(tpoly1.Equals(tpoly2));
}

```

```

    }

    [TestMethod]
    public void TestDiff1()
    {
        TPoly tpoly1 = new TPoly();
        tpoly1.Members.Add(new TMember(1, 3));

        string actual = "3x^2";

        Assert.AreEqual(tpoly1.Diff().Show(), actual);
    }

    [TestMethod]
    public void TestElementAt1()
    {
        TPoly tpoly1 = new TPoly();
        tpoly1.Members.Add(new TMember(11, 11));
        tpoly1.Members.Add(new TMember(22, 22));

        Assert.AreEqual(new System.Tuple<int, int>(11, 11),
            tpoly1.TakeElement(1));
    }

    [TestMethod]
    [ExpectedException(typeof(ArgumentOutOfRangeException))]
    public void TestElementAt2()
    {
        TPoly tpoly1 = new TPoly();
        tpoly1.Members.Add(new TMember(-21, 33));
        tpoly1.Members.Add(new TMember(6, -4));
        tpoly1.Members.Add(new TMember(10, 6));

        tpoly1.TakeElement(5);
    }

    [TestMethod]
    public void TestRetDegree()
    {
        TPoly tpoly1 = new TPoly();
        tpoly1.Members.Add(new TMember(11, 11));
        tpoly1.Members.Add(new TMember(22, 22));

        int actual = 22;

        Assert.AreEqual(tpoly1.ReturnDegree(), actual);
    }

    [TestMethod]
    public void TestRetCoeff1()
    {
        TPoly tpoly1 = new TPoly();
        tpoly1.Members.Add(new TMember(3, 7));
        tpoly1.Members.Add(new TMember(-84, 15));

        int actual = -84;

        Assert.AreEqual(tpoly1.ReturnCoefficient(15), actual);
    }

```

```

[TestMethod]
[ExpectedException(typeof(InvalidOperationException))]
public void TestRetCoeff2()
{
    TPoly tpoly1 = new TPoly();
    tpoly1.Members.Add(new TMember(-7, 32));
    tpoly1.Members.Add(new TMember(7, 6));
    tpoly1.Members.Add(new TMember(112, 2));

    tpoly1.ReturnCoefficient(33);
}

[TestMethod]
public void TestMinus()
{
    TPoly tpoly1 = new TPoly();
    tpoly1.Members.Add(new TMember(11, 1));
    tpoly1.Members.Add(new TMember(-22, 3));
    tpoly1.Members.Add(new TMember(20, 2));

    string actual = "22x^3-20x^2-11x^1";

    Assert.AreEqual(tpoly1.Minus().Show(), actual);
}
}

```