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”.**

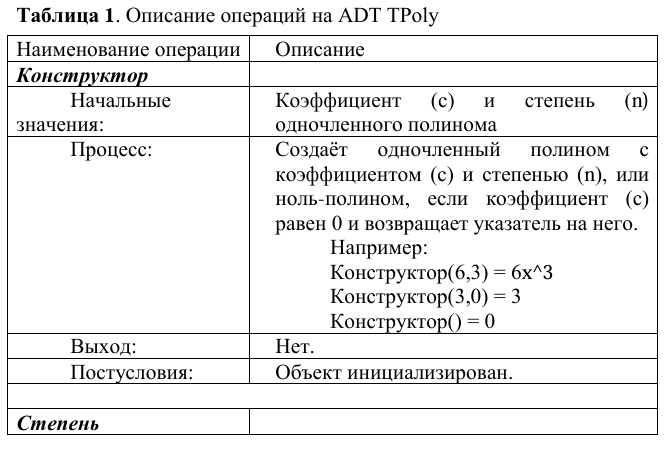
## ADT TPoly

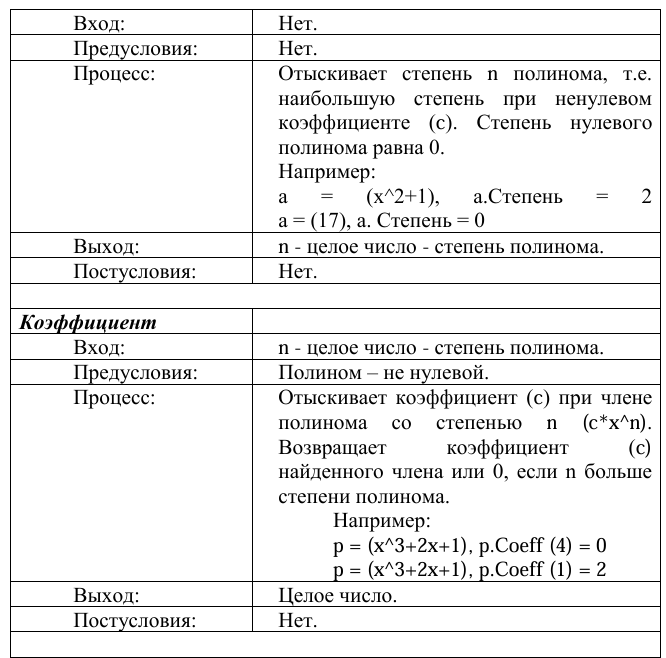
**Данные**

Полиномы Tpoly — это неизменяемые полиномы с целыми коэффициентами.

**Операции**

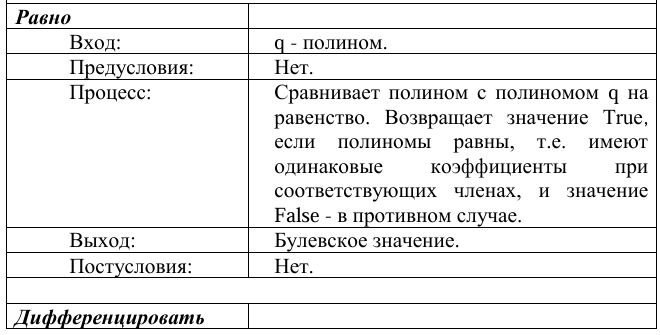
Операции могут вызываться только объектом «полином» (тип TPoly), указатель на который передаётся в них по умолчанию. При описании операций этот объект в разделе «Вход» не указывается.

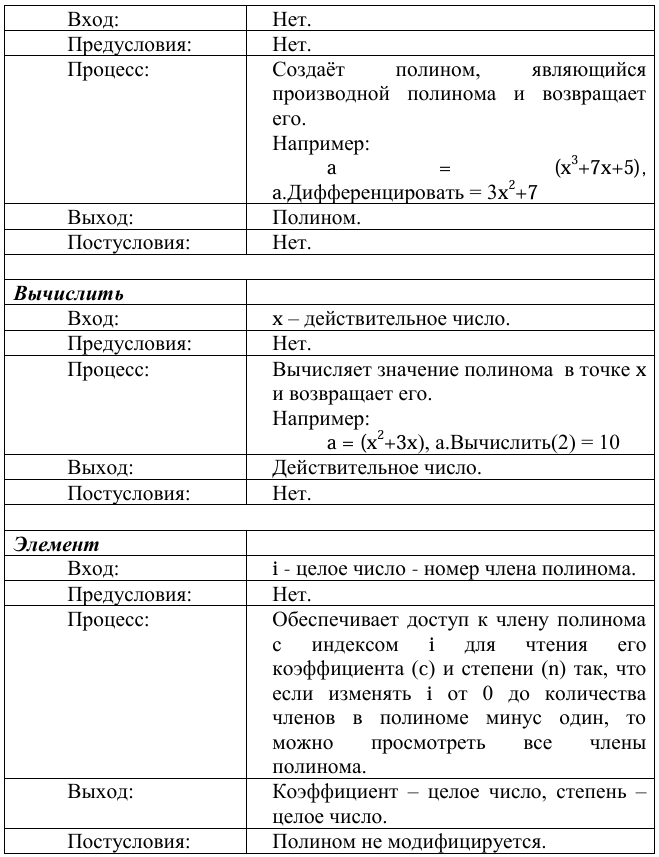












**Спецификация абстрактного типа данных “Одночлен”.**

## ADT TMember

**Данные**

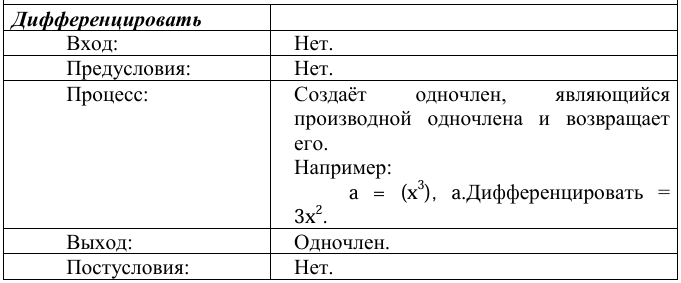
Одночлен TMember - это изменяемые одночленные полиномы с целыми коэффициентами. Коэффициент и степень хранятся в полях целого типа FCoeff и FDegree соответственно.

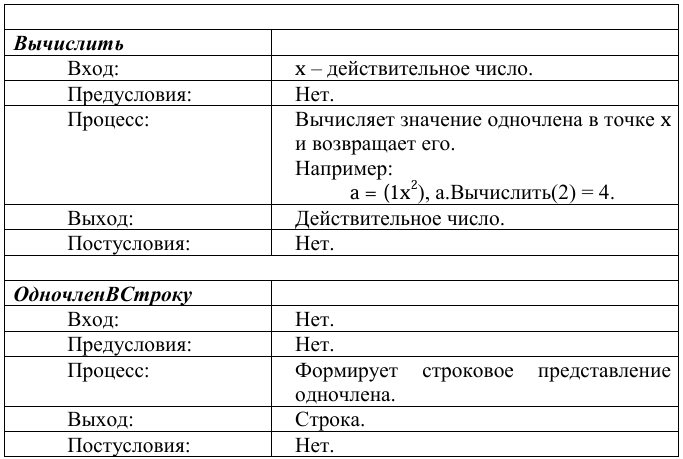
**Операции**

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.









# Реализация:

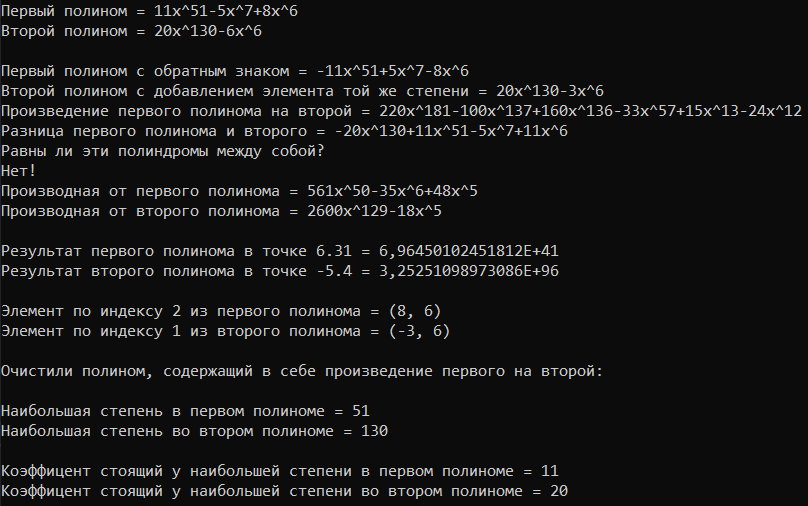


Рис. 1 – Результат проверки работоспособности программы.

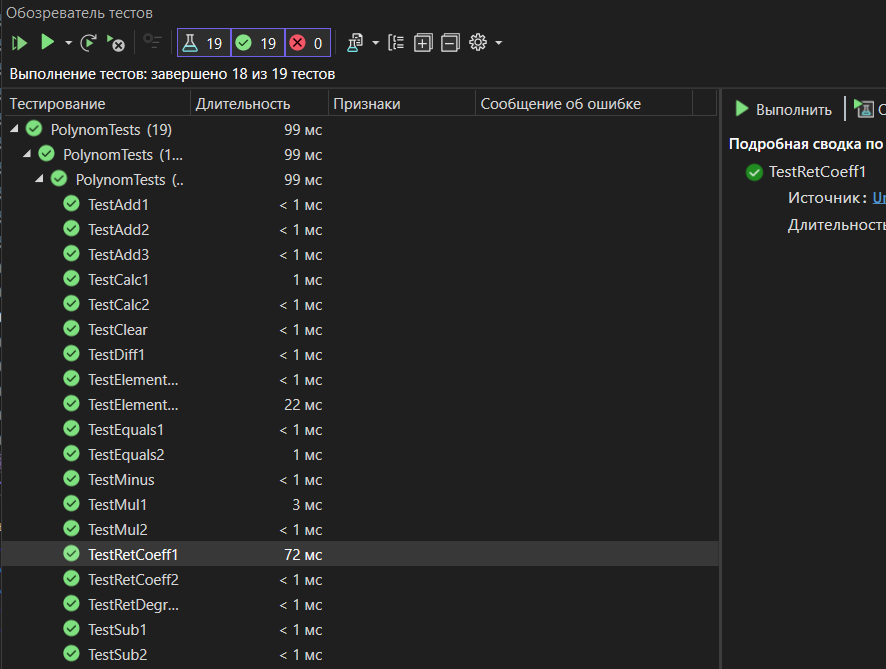


Рис. 2 – Результат выполнения модульных тестов.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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>.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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 противоложный.

# Вывод

В результате работы над лабораторной работой были сформированы практические навыки реализации параметризованного абстрактного типа данных на языке С#, разработки функций классов на языке С#, разработка модульных тестов для тестирования функций классов и выполнения модульного тестирования на языке С# 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.ReturnCoefficent(degreeFirst);

**int** maxCoeffSecond = tPoly2.ReturnCoefficent(degreeSecond);

Console.WriteLine($"Коэффицент стоящий у наибольшей степени в первом полиноме = {maxCoeffFirst}\nКоэффицент 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** \_fcoefficent;

**private** **int** \_fdegree;

**public** **int** FCoefficent

{

**get** { **return** \_fcoefficent; }

**set**

{

**if** (**value** == **0**)

{

\_fdegree = **0**;

}

\_fcoefficent = **value**;

}

}

**public** **int** FDegree

{

**get** { **return** \_fdegree; }

**set**

{

**if**(\_fcoefficent == **0**)

{

\_fdegree = **0**;

}

**else** \_fdegree = **value**;

}

}

**public** **TMember**(**int** coeff = **0**, **int** degree = **0**)

{

FCoefficent = coeff;

FDegree = degree;

}

**public** **override** **bool** **Equals**(**object** obj)

{

**if**((((TMember)obj).FCoefficent == **this**.FCoefficent) && (((TMember)obj).FDegree == **this**.FDegree))

**return** **true**;

**else**

**return** **false**;

}

**public** TMember **Differentiate**()

{

**return** **new** **TMember**()

{

FCoefficent = (**this**.FCoefficent \* **this**.FDegree),

FDegree = (**this**.FDegree - **1**)

};

}

**public** **double** **Calculate**(**double** a)

{

**return** **this**.FCoefficent \* Math.Pow(a, **this**.FDegree);

}

**public** **string** **TMemberToString**()

{

**return** (**this**.FCoefficent == **0**) ? "" : $"{this.FCoefficent}x^{this.FDegree}";

}

**public** **int** **CompareTo**(TMember other)

{

**if**(**this**.FDegree.CompareTo(other.FDegree) != **0**)

**return** **this**.FDegree.CompareTo(other.FDegree);

**else**

{

other.FCoefficent += **this**.FCoefficent;

**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.FCoefficent, m.FDegree));

}

**foreach**(TMember m **in** **this**.Members)

{

newPoly.Members.Add(**new** TMember(m.FCoefficent, 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.FCoefficent \* m.FCoefficent, newM.FDegree + m.FDegree));

}

}

**return** newPoly;

}

**public** TPoly **Sub**(TPoly other)

{

TPoly newPoly = **new** TPoly();

**foreach**(TMember m **in** other.Members)

{

newPoly.Members.Add(**new** TMember(-m.FCoefficent, m.FDegree));

}

**foreach**(TMember m **in** **this**.Members)

{

newPoly.Members.Add(**new** TMember(m.FCoefficent, m.FDegree));

}

**return** newPoly;

}

**public** TPoly **Minus**()

{

TPoly newPoly = **new** TPoly();

**foreach**(TMember m **in** **this**.Members)

{

newPoly.Members.Add(**new** TMember(-m.FCoefficent, m.FDegree));

}

**return** newPoly;

}

**public** **override** **bool** **Equals**(**object** obj)

{

**if** (((TPoly)obj).Members.SequenceEqual(**this**.Members))

**return** **true**;

**else**

**return** **false**;

}

**public** TPoly **Diff**()

{

TPoly newPoly = **new** TPoly();

**foreach**(TMember m **in** **this**.Members)

{

newPoly.Members.Add(**new** TMember(m.FCoefficent, 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).FCoefficent,

**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** **ReturnCoefficent**(**int** n)

{

**if**(n >= Members.First().FDegree && n <= Members.Last().FDegree)

{

**return** Members.Single(x => x.FDegree == n).FCoefficent;

}

**else**

{

**throw** **new** **InvalidOperationException**();

}

}

**public** **string** **Show**()

{

**string** str = "";

**foreach**(TMember m **in** **this**.Members.Reverse())

{

str += (m.FCoefficent > **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.ReturnCoefficent(**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.ReturnCoefficent(**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);

}

}

}