МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное

образовательное учреждение

высшего образования

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

**по дисциплине**

**“** **Тестирование программного обеспечения”**

**Инструмент IntelliTest**

Выполнил: студент гр. ФИб-4302-51-00 Шатунов Д. И. \_\_\_\_\_\_

Проверил: старший преподаватель кафедры ПМиИ Фищева И.Н. \_\_\_\_\_

Киров 2022

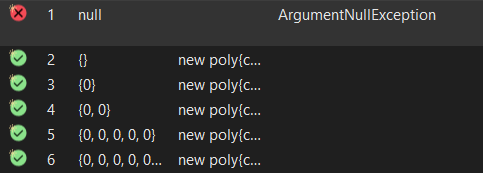
**Задание:** Вам необходимо взять программу на С#. В этой программе должно быть несколько классов, отвечающих за логику работы приложения.

Нас интересуют в первую очередь они, а не классы-формы.

С помощью инструментов IntelliTest создайте модульные тесты к классам программы.

Для задания был взят класс Polynom определяющий различные операции над многочленами.

**Найденные ошибки**

****

**Конструктор poly**

public poly(List<double> coef) // Инициализация списком

{

this.coef = new List<double>();

for (int i = 0; i < coef.Count(); i++)

this.coef.Add(coef[i]);

}

В данном конструкторе отсутствует проверка входного параметра coef на пустое значение.

Исправленный код:

public poly(List<double> coef) // Инициализация списком

{

if (coef == null)

throw new ArgumentNullException(nameof(coef));

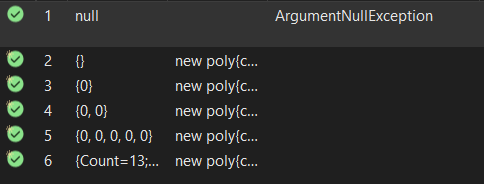
this.coef = new List<double>();

for (int i = 0; i < coef.Count(); i++)

this.coef.Add(coef[i]);

}

**Пройденные тесты**



**Конструктор poly 2**

public poly(string fileName, bool isFileInput, int num) // Инициализация значениями из файла

{

if (isFileInput == true)

{

this.coef = new List<double>();

using (StreamReader sr = new StreamReader(fileName, System.Text.Encoding.Default))

{

string line = File.ReadLines(fileName).Skip(num).First();

string[] eachCoef = line.Split(' ');

for (int i = 0; i < eachCoef.Length; i++)

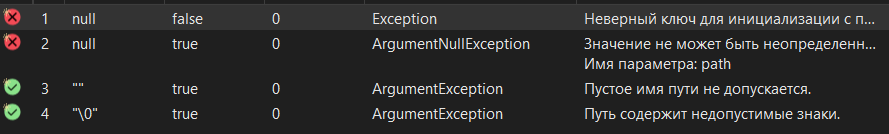
this.coef.Add(double.Parse(eachCoef[i]));

}

}

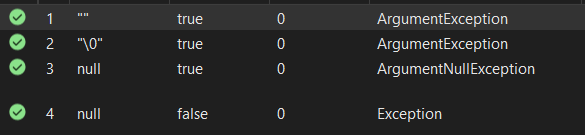
else throw new Exception("Неверный ключ для инициализации с помощью файла");

}



В данном случае первая ошибка является запланированным исключением программы, поэтому ее мы разрешаем, вторая ошибка появляется при условии, что значения инициализируются из файла, но путь к файлу отсутствует, нужно проверять путь файла на пустое значение.

**Пройденные тесты**



**Исправленный код** (добавлена проверка на null)

if (fileName== null)

throw new ArgumentNullException(nameof(fileName));

**Метод получения степени полинома**

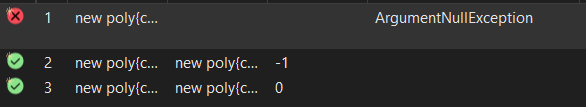
public int getDegree()

{

return coef.Count() - 1;

}

В данном методе также отсутствует проверка на пустое значение



Добавляем проверку

public int getDegree()

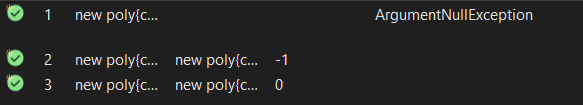
{

if (coef == null)

throw new ArgumentNullException(nameof(coef));

return coef.Count() - 1;

}



**Метод для вывода многочлена**

public void print() // Вывод в консоль в символьном виде

{

for (int i = coef.Count() - 1; i > 0; i--)

{

while (coef[i] == 0 && i > 0) i--; // Прохождение одночленов с нулевыми коэффициентами

if (i == 0)

{

break;

}

if (i == coef.Count() - 1) Console.Write("{0:0.##}", coef[i] + "x^" + i); // Нормализованный символьный вывод

else

if (coef[i] > 0) Console.Write("{0:0.##}", "+" + coef[i] + "x^" + i);

else Console.Write("{0:0.##}", coef[i] + "x^" + i);

}

if (coef.Count() == 1) Console.Write("{0:0.##}", coef[0]);

else

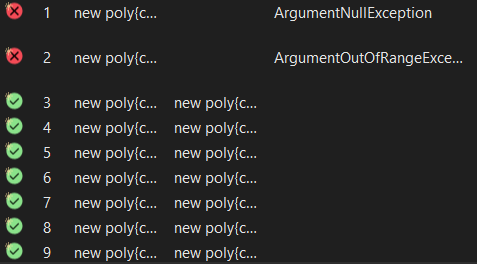
if (coef[0] > 0) Console.Write("{0:0.##}", "+" + coef[0]); // Вывод одночлена нулевой степени

else if (coef[0] < 0) Console.Write("{0:0.##}", coef[0]);

Console.WriteLine();

}

В данном случае в методе нет проверки является ли поле coef null или является ли оно пустым, что приводит к ошибкам



Добавление проверки

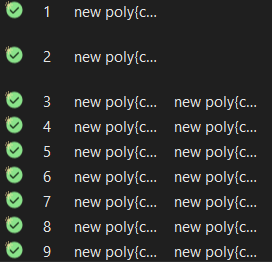
if (coef == null)

throw new ArgumentNullException(nameof(coef));

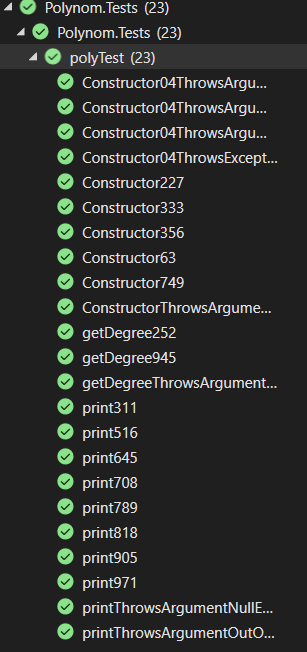
if (coef.Count == 0)

throw new ArgumentOutOfRangeException(nameof(coef));

Пройденные тесты



Все пройденные сгенерированные тесты



Вывод: IntelliTest можно применять в случае, когда программа имеет большое количество методов или когда эти методы сильно завязаны на других вручную созданных объектах, где есть вероятность ошибиться и не учесть возможные случаи работы программы. При написании программы я не задумывался о таких вариантах работы программы и не думал о их корректной обработке, найти ошибки удалось при помощи IntelliTest.

**Тесты**

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

[ExpectedException(typeof(ArgumentNullException))]

public void printThrowsArgumentNullException643()

{

poly poly;

poly = new poly();

poly.coef = (List<double>)null;

this.print(poly);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

[ExpectedException(typeof(ArgumentOutOfRangeException))]

public void printThrowsArgumentOutOfRangeException702()

{

List<double> list;

poly poly;

double[] ds = new double[0];

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

this.print(poly);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void print971()

{

List<double> list;

poly poly;

double[] ds = new double[1];

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

this.print(poly);

Assert.IsNotNull((object)poly);

Assert.IsNotNull((object)(poly.coef));

Assert.AreEqual<int>(1, poly.coef.Capacity);

Assert.AreEqual<int>(1, poly.coef.Count);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void print708()

{

List<double> list;

poly poly;

double[] ds = new double[2];

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

this.print(poly);

Assert.IsNotNull((object)poly);

Assert.IsNotNull((object)(poly.coef));

Assert.AreEqual<int>(2, poly.coef.Capacity);

Assert.AreEqual<int>(2, poly.coef.Count);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void print516()

{

List<double> list;

poly poly;

double[] ds = new double[2];

ds[0] = 1;

ds[1] = 2;

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

this.print(poly);

Assert.IsNotNull((object)poly);

Assert.IsNotNull((object)(poly.coef));

Assert.AreEqual<int>(2, poly.coef.Capacity);

Assert.AreEqual<int>(2, poly.coef.Count);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void print905()

{

List<double> list;

poly poly;

double[] ds = new double[3];

ds[0] = 1;

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

this.print(poly);

Assert.IsNotNull((object)poly);

Assert.IsNotNull((object)(poly.coef));

Assert.AreEqual<int>(3, poly.coef.Capacity);

Assert.AreEqual<int>(3, poly.coef.Count);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void print645()

{

List<double> list;

poly poly;

double[] ds = new double[2];

ds[0] = -38;

ds[1] = 2;

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

this.print(poly);

Assert.IsNotNull((object)poly);

Assert.IsNotNull((object)(poly.coef));

Assert.AreEqual<int>(2, poly.coef.Capacity);

Assert.AreEqual<int>(2, poly.coef.Count);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void print311()

{

List<double> list;

poly poly;

double[] ds = new double[11];

ds[0] = 2;

ds[1] = 2;

ds[2] = 2;

ds[3] = 2;

ds[4] = 2;

ds[5] = 2;

ds[6] = 2;

ds[7] = 2;

ds[8] = 4;

ds[9] = 1;

ds[10] = 3;

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

this.print(poly);

Assert.IsNotNull((object)poly);

Assert.IsNotNull((object)(poly.coef));

Assert.AreEqual<int>(11, poly.coef.Capacity);

Assert.AreEqual<int>(11, poly.coef.Count);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void print818()

{

List<double> list;

poly poly;

double[] ds = new double[11];

ds[0] = 1;

ds[1] = 1;

ds[2] = 1;

ds[3] = 1;

ds[4] = 1;

ds[5] = 1;

ds[6] = 1;

ds[7] = 1;

ds[8] = 3;

ds[9] = -1;

ds[10] = 2;

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

this.print(poly);

Assert.IsNotNull((object)poly);

Assert.IsNotNull((object)(poly.coef));

Assert.AreEqual<int>(11, poly.coef.Capacity);

Assert.AreEqual<int>(11, poly.coef.Count);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void print789()

{

List<double> list;

poly poly;

double[] ds = new double[12];

ds[0] = 1;

ds[1] = 1;

ds[2] = 1;

ds[3] = 1;

ds[4] = 1;

ds[5] = 1;

ds[6] = 1;

ds[7] = 1;

ds[8] = 1;

ds[9] = -1;

ds[10] = -39;

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

this.print(poly);

Assert.IsNotNull((object)poly);

Assert.IsNotNull((object)(poly.coef));

Assert.AreEqual<int>(12, poly.coef.Capacity);

Assert.AreEqual<int>(12, poly.coef.Count);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

[ExpectedException(typeof(ArgumentNullException))]

public void ConstructorThrowsArgumentNullException270()

{

poly poly;

poly = this.Constructor((List<double>)null);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void Constructor63()

{

List<double> list;

poly poly;

double[] ds = new double[0];

list = new List<double>((IEnumerable<double>)ds);

poly = this.Constructor(list);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void Constructor749()

{

List<double> list;

poly poly;

double[] ds = new double[1];

list = new List<double>((IEnumerable<double>)ds);

poly = this.Constructor(list);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void Constructor356()

{

List<double> list;

poly poly;

double[] ds = new double[2];

list = new List<double>((IEnumerable<double>)ds);

poly = this.Constructor(list);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void Constructor333()

{

List<double> list;

poly poly;

double[] ds = new double[5];

list = new List<double>((IEnumerable<double>)ds);

poly = this.Constructor(list);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void Constructor227()

{

List<double> list;

poly poly;

double[] ds = new double[13];

list = new List<double>((IEnumerable<double>)ds);

poly = this.Constructor(list);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

[ExpectedException(typeof(ArgumentException))]

public void Constructor04ThrowsArgumentException384()

{

poly poly;

poly = this.Constructor04("", true, 0);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

[ExpectedException(typeof(ArgumentException))]

public void Constructor04ThrowsArgumentException658()

{

poly poly;

poly = this.Constructor04("\0", true, 0);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

[ExpectedException(typeof(ArgumentNullException))]

public void Constructor04ThrowsArgumentNullException32()

{

poly poly;

poly = this.Constructor04((string)null, true, 0);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

[ExpectedException(typeof(Exception))]

public void Constructor04ThrowsException692()

{

poly poly;

poly = this.Constructor04((string)null, false, 0);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

[ExpectedException(typeof(ArgumentNullException))]

public void getDegreeThrowsArgumentNullException628()

{

poly poly;

int i;

poly = new poly();

poly.coef = (List<double>)null;

i = this.getDegree(poly);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void getDegree252()

{

List<double> list;

poly poly;

int i;

double[] ds = new double[0];

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

i = this.getDegree(poly);

Assert.IsNotNull((object)poly);

Assert.IsNotNull((object)(poly.coef));

Assert.AreEqual<int>(0, poly.coef.Capacity);

Assert.AreEqual<int>(0, poly.coef.Count);

}

[TestMethod]

[PexGeneratedBy(typeof(polyTest))]

public void getDegree945()

{

List<double> list;

poly poly;

int i;

double[] ds = new double[1];

list = new List<double>((IEnumerable<double>)ds);

poly = new poly();

poly.coef = list;

i = this.getDegree(poly);

Assert.IsNotNull((object)poly);

Assert.IsNotNull((object)(poly.coef));

Assert.AreEqual<int>(1, poly.coef.Capacity);

Assert.AreEqual<int>(1, poly.coef.Count);

}