МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №5 по дисциплине

“Коды, исправляющие ошибки”

Выполнил студент группы ПМИ-3301-02-00     / А.Л. Вохмянин /

Проверил преподаватель кафедры ПМИ     / И.А. Пушкарёв /

Киров 2019

**Цель:** реализовать программу, реализующую деление многочленов с остатком и представление в линейном виде.

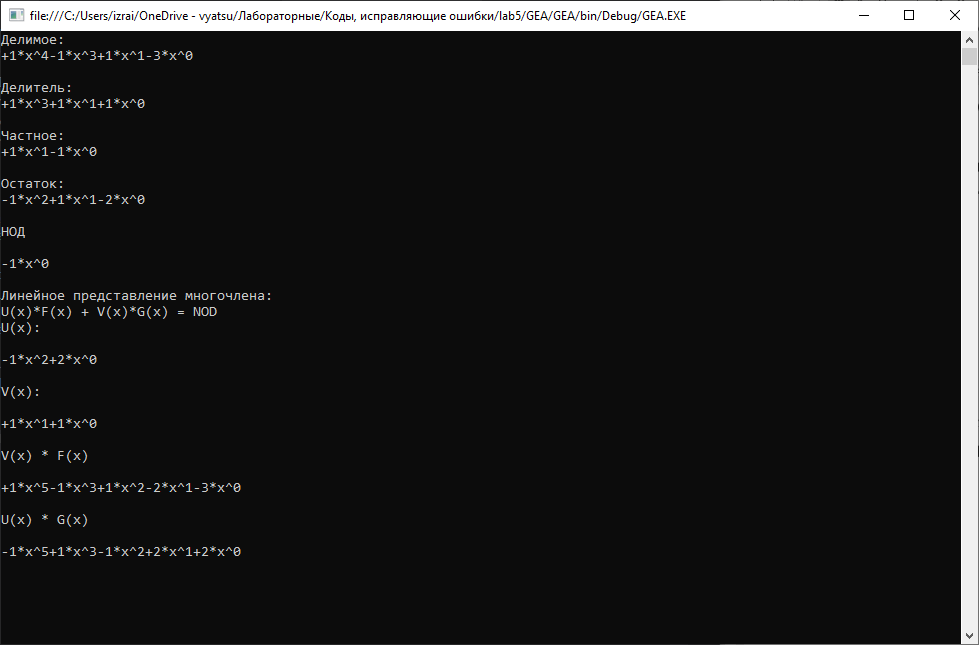
Пусть a и b — целые числа, не равные одновременно нулю, и последовательность чисел

определена тем, что каждое — это остаток от деления предпредыдущего числа на предыдущее, а предпоследнее делится на последнее нацело.

Тогда НОД(a, b), наибольший общий делитель a и b, равен , последнему ненулевому члену этой последовательности

Алгоритм Евклида и расширенный алгоритм Евклида естественным образом обобщается на [кольцо многочленов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D1%8C%D1%86%D0%BE_%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD%D0%BE%D0%B2) k[x] от одной переменной над произвольным [полем](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_(%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0)) k, поскольку для таких многочленов определена операция деления с остатком. При выполнении алгоритма Евклида для многочленов аналогично алгоритму Евклида для целых чисел получается последовательность полиномиальных остатков

**Результаты выполнения программы:**



**Вывод:** былиизучены и получены практические навыки использования алгоритма Евклида для многочленов.

**Листинг:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GEA

{

class Program

{

const double eps = 0.0001;

static List<double> NOD;

public static class MyMath

{

public static List<double> Minus(List<double> polynom1, List<double> polynom2)

{

int itemsCount = Math.Max(polynom1.Count, polynom2.Count);

var result = new List<double>(new double [itemsCount]);

for (int i = 0; i < itemsCount; i++)

{

double a = 0;

double b = 0;

if (i < polynom1.Count)

{

a = polynom1[i];

}

if (i < polynom2.Count)

{

b = polynom2[i];

}

result[i] = a - b;

}

return result;

}

public static List<double> Multiplication(List<double> polynom1, List<double> polynom2)

{

List<double> result = new List<double>(new double[polynom1.Count + polynom2.Count - 1]);

for (int i = 0; i < polynom1.Count; i++)

for (int j = 0; j < polynom2.Count; j++)

{

if (result[i + j] != 0)

{

result[i + j] += polynom1[i] \* polynom2[j];

}

else

result[i + j] = polynom1[i] \* polynom2[j];

}

return result;

}

public static List<double> Add(List<double> polynom1, List<double> polynom2)

{

int count = Math.Max(polynom1.Count, polynom2.Count);

List<double> result = new List<double>(new double[count]);

for (int i = 0; i < count; i++)

{

double a = 0;

double b = 0;

if (i < polynom1.Count)

{

a = polynom1[i];

}

if (i < polynom2.Count)

{

b = polynom2[i];

}

result[i] = a + b;

}

return result;

}

public static void Deconv(List<double> dividend, List<double> divisor, out List<double> quotient, out List<double> remainder)

{

if (dividend.Count == 0)

{

remainder = new List<double>(dividend);

quotient = new List<double>(new double[remainder.Count - divisor.Count + 1]);

return;

}

if (divisor.Count == 0)

{

remainder = new List<double>(dividend);

quotient = new List<double>(new double[remainder.Count - divisor.Count + 1]);

return;

}

while (dividend.Last() == 0)

{

dividend.RemoveAt(dividend.Count - 1);

if (dividend.Count == 0)

{

remainder = new List<double>(dividend);

quotient = new List<double>(new double[remainder.Count - divisor.Count + 1]);

return;

}

}

while (divisor.Last() == 0)

{

divisor.RemoveAt(divisor.Count - 1);

if (divisor.Count == 0)

{

remainder = new List<double>(dividend);

quotient = new List<double>(new double[remainder.Count - divisor.Count + 1]);

return;

}

}

remainder = new List<double>(dividend);

quotient = new List<double>(new double[remainder.Count - divisor.Count + 1]);

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

{

double coeff = remainder[remainder.Count - i - 1] / divisor.Last();

quotient[quotient.Count - i - 1] = coeff;

for (int j = 0; j < divisor.Count; j++)

{

remainder[remainder.Count - i - j - 1] -= coeff \* divisor[divisor.Count - j - 1];

}

}

}

public static bool Equal(List<double> polynom1, List<double> polynom2)

{

int count = 0;

if (polynom1.Count == polynom2.Count)

{

for (int i = 0; i < polynom1.Count; i++)

{

if (polynom1[i] == polynom2[i])

{

count++;

}

}

if (count == polynom1.Count)

return true;

}

return false;

}

public static List<double> gcd(List<double> a, List<double> b, out List<double> x, out List<double> y)

{

if (a.Count == 0)

{

x = new List<double> { 0 };

y = new List<double> { 1 };

return b;

}

List<double> x1, y1;

List<double> quotient;

List<double> remainder;

MyMath.Deconv(b, a, out quotient, out remainder);

List<double> d = gcd(remainder, a, out x1, out y1);

x = Minus(y1, (Multiplication(quotient, x1)));

y = new List<double>(x1);

return d;

}

}

public static void Print(List<double> pol)

{

Console.WriteLine();

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

{

if (pol[pol.Count - i - 1] != 0)

{

Console.Write("{0}{1}\*x^{2}", pol[pol.Count - i - 1] >= 0 ? "+" : "", pol[pol.Count - i - 1], pol.Count - i - 1);

}

}

Console.WriteLine("\n");

}

// деление многочлена с остатком

// представить в виде умножения + остаток

// Необходимо найти их НОД и представить его в линейном виде

static void Main(string[] args)

{

List<double> dividend = new List<double> {-3, 1, 0, -1, 1};

List<double> divisor = new List<double>{1, 1, 0, 1};

List<double> quotient;

List<double> remainder;

#region Вывод dividend = divisor \* quotient + remainder

MyMath.Deconv(dividend, divisor, out quotient, out remainder);

Console.Write("Делимое:");

Print(dividend);

Console.Write("Делитель:");

Print(divisor);

Console.Write("Частное:");

Print(quotient);

Console.Write("Остаток:");

Print(remainder);

#endregion

List<double> tempDividend = new List<double> { -3, 1, 0, -1, 1 };

List<double> tempDivisor = new List<double> { 1, 1, 0, 1 };

List<double> temp = new List<double>(remainder);

bool flag = false;

while (true)

{

while (remainder.Last() == 0 || Math.Abs(remainder.Last()) < eps)

{

remainder.RemoveAt(remainder.Count - 1);

if (remainder.Count == 0)

{

remainder = new List<double>(temp);

flag = true;

break;

}

}

if (flag == true)

break;

tempDividend = new List<double>(tempDivisor);

tempDivisor = new List<double>(remainder);

temp = new List<double>(remainder);

MyMath.Deconv(tempDividend, tempDivisor, out quotient, out remainder);

}

NOD = new List<double>(remainder);

Console.WriteLine("НОД");

Print(NOD);

List<double> V = new List<double>();

List<double> U = new List<double> {};

MyMath.gcd(dividend, divisor, out V, out U);

Console.WriteLine("Линейное представление многочлена: \nU(x)\*F(x) + V(x)\*G(x) = NOD");

//Console.WriteLine("F(x):");

//Print(dividend);

//Console.WriteLine("G(x):");

//Print(divisor);

Console.WriteLine("U(x):");

Print(U);

Console.WriteLine("V(x):");

Print(V);

Console.WriteLine("V(x) \* F(x)");

Print(MyMath.Multiplication(V, dividend));

Console.WriteLine("U(x) \* G(x)");

Print(MyMath.Multiplication(U, divisor));

Console.ReadKey();

}

}

}