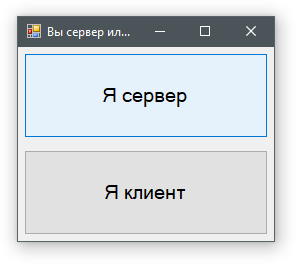
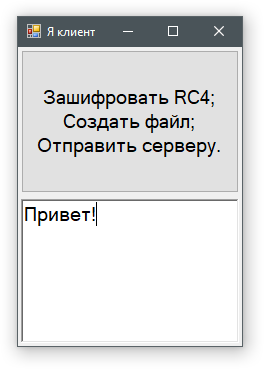
**Лабораторная работа №4. RC4. Передача файла.**

**Десятов Александр,**

**09-641**





using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Numerics;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace RC4\_передача\_файла

{

public partial class FormClient : Form

{

string file\_path\_client = Environment.CurrentDirectory + @"/client/text.bin"; //

string file\_path\_server = Environment.CurrentDirectory + @"/server/text.bin"; //

static Socket socket\_client\_getting = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

static Socket socket\_client\_sending = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

Socket socket\_from\_server;

int numeric\_len\_b = 128; // default

Random rnd;

BigInteger K;

#region Form

public FormClient()

{

InitializeComponent();

}

private void FormClient\_Shown(object sender, EventArgs e)

{

rnd = new Random((int)DateTime.Now.Ticks);

// Запуск фонового потока ПОЛУЧЕНИЕ АСИНХРОННОГО КЛЮЧА K

bW\_KeysExchange.RunWorkerAsync();

}

private void FormClient\_FormClosing(object sender, FormClosingEventArgs e)

{

if (bW\_KeysExchange.IsBusy)

{

bW\_KeysExchange.WorkerSupportsCancellation = true;

bW\_KeysExchange.CancelAsync();

}

if (bW\_code\_createFile\_send.IsBusy)

{

bW\_code\_createFile\_send.WorkerSupportsCancellation = true;

bW\_code\_createFile\_send.CancelAsync();

}

socket\_client\_getting.Close();

socket\_client\_sending.Close();

//socket\_from\_server.Close();

}

// Поток обмена ключами

private void bW\_KeysExchange\_DoWork(object sender, DoWorkEventArgs e)

{

socket\_client\_getting.Bind(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 905));//Привязка

socket\_client\_getting.Listen(1); // Максимальная очередь

socket\_from\_server = socket\_client\_getting.Accept(); // Принимает подключение

byte[] buffer = new byte[1024]; // ???

socket\_from\_server.Receive(buffer); // Принимает p,g,A в буфер

//... bufer to...p, g, A

int len\_p = BitConverter.ToInt32(buffer, 0);

int len\_g = BitConverter.ToInt32(buffer, sizeof(int));

int len\_A = BitConverter.ToInt32(buffer, 2 \* sizeof(int));

int beg\_p = 3 \* sizeof(int);

int beg\_g = 3 \* sizeof(int) + len\_p;

int beg\_A = 3 \* sizeof(int) + len\_p + len\_g;

byte[] p\_b = new byte[len\_p];

for (int i = 0; i < len\_p; i++)

p\_b[i] = buffer[i + beg\_p];

BigInteger p = new BigInteger(p\_b);

byte[] g\_b = new byte[len\_g];

for (int i = 0; i < len\_g; i++)

g\_b[i] = buffer[i + beg\_g];

BigInteger g = new BigInteger(g\_b);

byte[] A\_b = new byte[len\_A];

for (int i = 0; i < len\_A; i++)

A\_b[i] = buffer[i + beg\_A];

BigInteger A = new BigInteger(A\_b);

// Генерация любого b

BigInteger b = SgenerirovatAny(numeric\_len\_b);

// Вычисление K и B

BigInteger B = Vozvedenie\_v\_step\_po\_mod(g, b, p);

K = Vozvedenie\_v\_step\_po\_mod(A, b, p);

// Пытаемся подключиться к серверу

while (!socket\_client\_sending.Connected)

{

try

{

socket\_client\_sending.Connect("127.0.0.1", 904);

}

catch {; }

Thread.Sleep(100);

}

// Отправка B

buffer = B.ToByteArray();

socket\_client\_sending.Send(buffer);

// У НАС ЕСТЬ АСИНХРОННЫЙ КЛЮЧ K

}

private void bW\_KeysExchange\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

// После обмена ключами, кнопка становится активной

btn\_code\_createFile\_send.Enabled = true;

}

private void btn\_code\_createFile\_send\_Click(object sender, EventArgs e)

{

bW\_code\_createFile\_send.RunWorkerAsync();

}

// Поток для шифрования, создания и отправки файла

private void bW\_code\_createFile\_send\_DoWork(object sender, DoWorkEventArgs e)

{

// Зашифровка RC4

byte[] codeRC4 = RC4(rTB\_text.Text, K);

File.Delete(file\_path\_client);

// Запись в бинарный файл

using (BinaryWriter writer = new BinaryWriter(File.Open(file\_path\_client, FileMode.Create)))

{

// запись массива байтов в файл

writer.Write(codeRC4, 0, codeRC4.Length);

}

// Отправка файла

SendFile(file\_path\_client, "127.0.0.1", 906);

}

private void SendFile(string file\_path, string ip, int port)

{

string IPAddress = ip;

int Port = port;

string Filename = file\_path;

int bufferSize = 1024;

FileStream fs = new FileStream(Filename, FileMode.Open);

int bufferCount = Convert.ToInt32(Math.Ceiling((double)fs.Length / (double)bufferSize));

TcpClient tcpClient = new TcpClient(IPAddress, Port);

tcpClient.SendTimeout = 600000;

tcpClient.ReceiveTimeout = 600000;

// Сразу говорим куда записать на сервере, по-хорошему, это переделать надо

string headerStr = "Content-length:" + fs.Length.ToString() + "\r\nFilename:" + file\_path\_server + "\r\n";

// Чтобы передать любой файл (а не только бинарный)

//string headerStr = "Content-length:" + fs.Length.ToString() + "\r\nFilename:" + file\_path\_client.Replace("client","server") + "\r\n";

byte[] header = new byte[bufferSize];

Array.Copy(Encoding.Default.GetBytes(headerStr), header, Encoding.Default.GetBytes(headerStr).Length);

tcpClient.Client.Send(header);

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

{

byte[] buffer = new byte[bufferSize];

int size = fs.Read(buffer, 0, bufferSize);

tcpClient.Client.Send(buffer, size, SocketFlags.Partial);

}

tcpClient.Client.Close();

fs.Close();

}

private byte[] RC4(string text, BigInteger k)

{

return RC4(Encoding.Default.GetBytes(text), k.ToByteArray());

}

private byte[] RC4(byte[] text, byte[] key)

{

byte[] answer = new byte[text.Length];

// Инициализация S блока

byte[] S = new byte[256];

for (int i = 0; i < 256; i++)

S[i] = (byte)i;

for (int i = 0, j = 0; i < 256; i++)

{

j = (j + S[i] + key[i % key.Length]) % 256;

Swap(ref S[i], ref S[j]);

}

// Конец инициализации S блока

for (int pos=0, i=0, j=0; pos<text.Length; pos++)

{

// Генерация pseudo\_random\_K

i = (i + 1) % 256;

j = (j + S[i]) % 256;

Swap(ref S[i], ref S[j]);

byte pseudo\_random\_K = S[(S[i] + S[j]) % 256];

answer[pos] = (byte) (text[pos] ^ pseudo\_random\_K);

}

return answer;

}

private void Swap<T>(ref T a, ref T b)

{

dynamic tmp = a;

a = b;

b = tmp;

}

#endregion

#region Генерация, Нод, Возведеине

// Генерация любого числа

private BigInteger SgenerirovatAny(int numeric\_len)

{// Получаем рандомное бинарное число длины numeric\_len и переводим сразу его в десятичное

BigInteger chislo = 1; // Первая цифра 1. Иначе для перевода двоичного в десятичное здесь был бы 0

while (numeric\_len-- > 1) // сначала сравнивает, потом вычитает

{

chislo <<= 1; //умножаем на 2

chislo += rnd.Next(2);

}

return chislo;

}

private BigInteger Vozvedenie\_v\_step\_po\_mod(BigInteger A, BigInteger B, BigInteger N)

{

#region Обработка нештатных ситуаций

if (N == 0)

return 0;

if (B == 0)

return 1;

if (A == 0)

return 0;

if (B < 0)

{

Find\_NOD\_X\_Y\_Euclid(N, A, out BigInteger nod, out BigInteger x, out BigInteger y);

A = y; // Так возвели A в (-1) степень, теперь надо в степень abs(B)

B = BigInteger.Abs(B);

}

#endregion

Stack<bool> bin = new Stack<bool>();

while (B > 1)

{

bin.Push(!B.IsEven); // B.IsEven == true, если четное, иначе false

B >>= 1; //Делим на 2 побитовым сдвигом

}

BigInteger answer = A;

while (bin.Count > 0)

{

if (bin.Pop())

answer = (answer \* answer \* A) % N;

else

answer = (answer \* answer) % N;

}

while (answer < 0)

answer += N;

return answer;

#region 2 метод из методчки

/\*BigInteger c = 1;

while(B != 0)

{

if(B%2 == 0)

{

B /= 2;

A = (A\*A)%N;

}

else

{

B--;

c=(c\*A)%N;

}

}

return c;\*/

#endregion

}

// НОД, x, y

private void Find\_NOD\_X\_Y\_Euclid(BigInteger A, BigInteger B, out BigInteger nod, out BigInteger x, out BigInteger y)

{

// Доллжно быть до любого return, так как out

x = y = nod = 0;

if (A == 0 || B == 0)

return;

// Чтобы запомнить частные от делений и посчитать потом x и y

Stack<BigInteger> whole\_stack = new Stack<BigInteger>();

// DivRem() - возвращает частное и через out предоставляет остаток от деления

BigInteger whole = BigInteger.DivRem(A, B, out BigInteger ostatok);

while (ostatok != 0)

{

whole\_stack.Push(whole);

A = B;

B = ostatok;

whole = BigInteger.DivRem(A, B, out ostatok);

}

nod = B; //BigInteger.Abs(B);

// Считаем x и y

x = 0;

y = 1;

while (whole\_stack.Count > 0)

{

BigInteger temp = x;

x = y;

y = (temp - y \* whole\_stack.Pop());

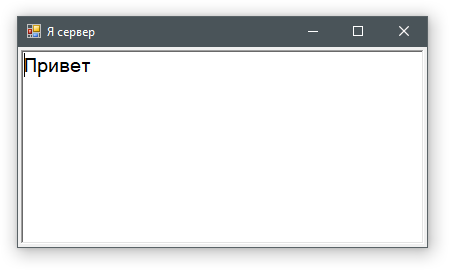
}

}

#endregion

}

}



using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net.Sockets;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Numerics;

using System.Threading;

using System.IO;

namespace RC4\_передача\_файла

{

public partial class FormServer : Form

{

string file\_path\_server = Environment.CurrentDirectory + @"/server/text.bin";

Socket socket\_server\_sending = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

Socket socket\_server\_getting = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

Socket socket\_from\_client;

int numeric\_len\_a = 128; // default

int numeric\_len\_p = 128; // default

int numeric\_len\_g = 128; // default

Random rnd;

BigInteger K;

#region Form

public FormServer()

{

InitializeComponent();

}

private void FormServer\_Shown(object sender, EventArgs e)

{

rnd = new Random((int)DateTime.Now.Ticks);

// Запуск фонового потока ПОЛУЧЕНИЕ АСИНХРОННОГО КЛЮЧА K

bW\_KeysExchange.RunWorkerAsync();

}

private void FormServer\_FormClosing(object sender, FormClosingEventArgs e)

{

if (bW\_KeysExchange.IsBusy)

{

bW\_KeysExchange.WorkerSupportsCancellation = true;

bW\_KeysExchange.CancelAsync();

}

if (bW\_waitFile.IsBusy)

{

bW\_waitFile.WorkerSupportsCancellation = true;

bW\_waitFile.CancelAsync();

}

socket\_server\_getting.Close();

socket\_server\_sending.Close();

//socket\_from\_client.Close();

}

// Поток обмена ключами

private void bW\_KeysExchange\_DoWork(object sender, DoWorkEventArgs e)

{

// генерирация любого a, g и p

BigInteger a = SgenerirovatAny(numeric\_len\_a);

BigInteger p = Sgenerirovat(numeric\_len\_p, 0);

// g отличное от p

BigInteger g = Sgenerirovat(numeric\_len\_g, p, true);

// подсчет A

BigInteger A = Vozvedenie\_v\_step\_po\_mod(g, a, p);

// p, g, A в массив байт

byte[] bytes\_p = p.ToByteArray();

byte[] bytes\_g = g.ToByteArray();

byte[] bytes\_A = A.ToByteArray();

// В один массив байт, чтобы потмо можно было считать

// сначала длины, потом значения

List<byte> listPaket = new List<byte>(bytes\_p.Length + bytes\_g.Length + bytes\_A.Length + 3 \* sizeof(int));

listPaket.AddRange(BitConverter.GetBytes(bytes\_p.Length));

listPaket.AddRange(BitConverter.GetBytes(bytes\_g.Length));

listPaket.AddRange(BitConverter.GetBytes(bytes\_A.Length));

listPaket.AddRange(bytes\_p);

listPaket.AddRange(bytes\_g);

listPaket.AddRange(bytes\_A);

byte[] buffer = listPaket.ToArray();

// Пытается подключиться к клиенту

while (!socket\_server\_sending.Connected)

{

try

{

socket\_server\_sending.Connect("127.0.0.1", 905);

}

catch {; }

Thread.Sleep(100);

}

// Отправка p,g,A

socket\_server\_sending.Send(buffer);

// Ждём B

socket\_server\_getting.Bind(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 904));//Привязка

socket\_server\_getting.Listen(1); // Максимальная очередь

socket\_from\_client = socket\_server\_getting.Accept(); // Принимает подключение

buffer = new byte[1024]; // ???

socket\_from\_client.Receive(buffer); // Принимает B в буфер

BigInteger B = new BigInteger(buffer);

K = Vozvedenie\_v\_step\_po\_mod(B, a, p);

// У НАС ЕСТЬ АСИНХРОННЫЙ КЛЮЧ K

}

private void bW\_KeysExchange\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

// Запуск потока ожидания файла

bW\_waitFile.RunWorkerAsync();

}

// Поток ожидания файла

private void bW\_waitFile\_DoWork(object sender, DoWorkEventArgs e)

{

//while (true)

// Получение бинарного файла

GetFile("127.0.0.1", 906);

// Чтение полученного бинарного файла

byte[] fileBytes = File.ReadAllBytes(file\_path\_server);

// Дешифровка RC4

byte[] textByte = RC4(fileBytes, K.ToByteArray());

string text = Encoding.Default.GetString(textByte);

// Вывод текста на экран

rTB\_text.Text = text;

}

private void GetFile(string ip, int port)

{

// Получение файла

int Port = port;

// C любого IP

//TcpListener listener = new TcpListener(IPAddress.Any, Port);

TcpListener listener = new TcpListener(IPAddress.Parse(ip), Port);

listener.Start();

Socket socket = listener.AcceptSocket();

int bufferSize = 1024;

byte[] header = new byte[bufferSize];

socket.Receive(header);

string headerStr = Encoding.Default.GetString(header);

string[] splitted = headerStr.Split(new string[] { "\r\n" }, StringSplitOptions.None);

Dictionary<string, string> headers = new Dictionary<string, string>();

foreach (string s in splitted)

{

if (s.Contains(":"))

{

headers.Add(s.Substring(0, s.IndexOf(":")), s.Substring(s.IndexOf(":") + 1));

}

}

//Get filesize from header

int filesize = Convert.ToInt32(headers["Content-length"]);

//Get filename from header

string filename = headers["Filename"];

//int bufferCount = Convert.ToInt32(Math.Ceiling((double)filesize / (double)bufferSize));

File.Delete(filename);

FileStream fs = new FileStream(filename, FileMode.Create);

while (filesize > 0)

{

byte[] buffer = new byte[bufferSize];

int size = socket.Receive(buffer, SocketFlags.Partial);

fs.Write(buffer, 0, size);

filesize -= size;

}

fs.Close();

}

private byte[] RC4(byte[] text, byte[] key)

{

byte[] answer = new byte[text.Length];

// Инициализация S блока

byte[] S = new byte[256];

for (int i = 0; i < 256; i++)

S[i] = (byte)i;

for (int i = 0, j = 0; i < 256; i++)

{

j = (j + S[i] + key[i % key.Length]) % 256;

Swap(ref S[i], ref S[j]);

}

// Конец инициализации S блока

for (int pos = 0, i = 0, j = 0; pos < text.Length; pos++)

{

// Генерация pseudo\_random\_K

i = (i + 1) % 256;

j = (j + S[i]) % 256;

Swap(ref S[i], ref S[j]);

byte pseudo\_random\_K = S[(S[i] + S[j]) % 256];

answer[pos] = (byte)(text[pos] ^ pseudo\_random\_K);

}

return answer;

}

private void Swap<T>(ref T a, ref T b)

{

dynamic tmp = a;

a = b;

b = tmp;

}

#endregion

#region Генерация чисел

// Генерация любого числа

private BigInteger SgenerirovatAny(int numeric\_len)

{// Получаем рандомное бинарное число длины numeric\_len и переводим сразу его в десятичное

BigInteger chislo = 1; // Первая цифра 1. Иначе для перевода двоичного в десятичное здесь был бы 0

while (numeric\_len-- > 1) // сначала сравнивает, потом вычитает

{

chislo <<= 1; //умножаем на 2

chislo += rnd.Next(2);

}

return chislo;

}

// Генерация вероятно простого числа и, если нужно, отличного от другого

private BigInteger Sgenerirovat(int numeric\_len, BigInteger first\_number, bool compare\_with = false)

{// Получаем рандомное бинарное число длины numeric\_len и сразу переделываем его в десятичное число

// И если compare\_with = true, то возвращается число отличное от first\_number

Random rnd = new Random((int)DateTime.Now.Ticks);

BigInteger chislo = 1; // Первая цифра 1. Иначе для перевода двоичного в десятичное здесь был бы 0

while (numeric\_len-- > 1) // сначала сравнивает, потом вычитает

{

chislo <<= 1; //умножаем на 2

chislo += rnd.Next(2);

}

if (chislo.IsEven)// Если четное

chislo += 1;

// Пока составное

// И пока не отличается от first\_number

while (Test\_Soloveya\_Shtrassena(chislo) == 0 || (compare\_with && chislo == first\_number))

chislo += 2;

return chislo; // Возвращаем вероятно простое число

}

// Получить рандомное BigInteger в диапазоне

private BigInteger GetRandomBigInteger(BigInteger min, BigInteger max)

{

if (min > max)

{

var buff = min;

min = max;

max = buff;

}

max -= min;

int deg = (int)BigInteger.Log(max, 2);

// Рандомная длина

deg = rnd.Next(deg + 1);

BigInteger tmp = 1;

while (deg-- > 0)

{

tmp <<= 1; //умножаем на 2

tmp += rnd.Next(2);

}

// Если вдруг больше max получилось

BigInteger.DivRem(tmp, max, out BigInteger result);

return result + min;

}

#endregion

#region Тесты на простоту

private int Test\_Millera\_Rabina(BigInteger n)

{

if (n < 2)

return -1; // не простое и не составное

else

{

int s = 0;

BigInteger d = n - 1;

while (d.IsEven)

{

d /= 2;

s++;

}

BigInteger[] prime\_numbers\_50 = {

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,

73, 79, 83, 89, 97, 101, /\*103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,

179, 181, 191, 193, 197, 199, 211, 223, 227, 229\*/};

// Сколько раз проверять?

BigInteger r = 50; // log2(n)

// Если число n-2 меньше, чем r, то нужно проверять не r раз, а (n-2) раз

foreach (BigInteger a in prime\_numbers\_50)

//r = n - 2 < r ? n - 2 : r;

//for (BigInteger a = 2; a <= r + 1; a++) // a должно по простым ходить...

{

if (a > n - 1)

break;

BigInteger x\_current = Vozvedenie\_v\_step\_po\_mod(a, d, n);

if (x\_current == 1 || x\_current == n - 1)

continue; // a - свидетель простоты

else

{

bool svidetel\_neprostoti = true;

for (int i = 1; i < s; i++)

{

x\_current = Vozvedenie\_v\_step\_po\_mod(x\_current, 2, n);

if (x\_current == n - 1)

{

svidetel\_neprostoti = false;

break; // a - свидетель простоты

}

}

if (svidetel\_neprostoti)

return 0; // составное

}

}

return 1; // вероятно простое

}

}

private int Test\_Soloveya\_Shtrassena(BigInteger n)

{

if (n < 2)

return -1; // не простое и не составное

else if (n == 2)

return 1; // простое

else if (n.IsEven)

return 0; // составное

int k = 10;

while (k-- > 0)

{

// Случайным образом выбирается число a<n

BigInteger a = GetRandomBigInteger(2, n);

if (NOD\_Euclid(a, n) > 1)

{

return 0; // Составное

}

else

{

BigInteger res1 = Vozvedenie\_v\_step\_po\_mod(a, (n - 1) / 2, n);

BigInteger res2 = Yakobi\_symbol(a, n);

while (res2 < 0)

res2 += n;

if (res1 != res2)

return 0; // Составное

}

}

return 1; // Вероятно простое

}

#endregion

#region Символ Якоби и Возведение в степень по модулю

private int Yakobi\_symbol(BigInteger a, BigInteger b)

{

if (NOD\_Euclid(a, b) != 1)

return 0;

int r = 1;

if (a < 0)

{

a = -a;

if (b % 4 == 3)

r = -r;

}

while (a != 0)

{

BigInteger t = 0;

while (a.IsEven)

{

t++;

a >>= 1; //деление на 2

}

if (!t.IsEven)

{

BigInteger.DivRem(b, 8, out BigInteger ostatok\_b\_na\_8);

if (ostatok\_b\_na\_8 == 3 || ostatok\_b\_na\_8 == 5)

r = -r;

}

BigInteger.DivRem(a, 4, out BigInteger ostatok\_a\_na\_4);

BigInteger.DivRem(b, 4, out BigInteger ostatok\_b\_na\_4);

if (ostatok\_a\_na\_4 == ostatok\_b\_na\_4 && ostatok\_a\_na\_4 == 3)

r = -r;

BigInteger c = a;

BigInteger.DivRem(b, c, out a);

b = c;

}

return r;

}

private BigInteger Vozvedenie\_v\_step\_po\_mod(BigInteger A, BigInteger B, BigInteger N)

{

#region Обработка нештатных ситуаций

if (N == 0)

return 0;

if (B == 0)

return 1;

if (A == 0)

return 0;

if (B < 0)

{

Find\_NOD\_X\_Y\_Euclid(N, A, out BigInteger nod, out BigInteger x, out BigInteger y);

A = y; // Так возвели A в (-1) степень, теперь надо в степень abs(B)

B = BigInteger.Abs(B);

}

#endregion

Stack<bool> bin = new Stack<bool>();

while (B > 1)

{

bin.Push(!B.IsEven); // B.IsEven == true, если четное, иначе false

B >>= 1; //Делим на 2 побитовым сдвигом

}

BigInteger answer = A;

while (bin.Count > 0)

{

if (bin.Pop())

answer = (answer \* answer \* A) % N;

else

answer = (answer \* answer) % N;

}

while (answer < 0)

answer += N;

return answer;

#region 2 метод из методчки

/\*BigInteger c = 1;

while(B != 0)

{

if(B%2 == 0)

{

B /= 2;

A = (A\*A)%N;

}

else

{

B--;

c=(c\*A)%N;

}

}

return c;\*/

#endregion

}

#endregion

#region НОД

// НОД, x, y

private void Find\_NOD\_X\_Y\_Euclid(BigInteger A, BigInteger B, out BigInteger nod, out BigInteger x, out BigInteger y)

{

// Доллжно быть до любого return, так как out

x = y = nod = 0;

if (A == 0 || B == 0)

return;

// Чтобы запомнить частные от делений и посчитать потом x и y

Stack<BigInteger> whole\_stack = new Stack<BigInteger>();

// DivRem() - возвращает частное и через out предоставляет остаток от деления

BigInteger whole = BigInteger.DivRem(A, B, out BigInteger ostatok);

while (ostatok != 0)

{

whole\_stack.Push(whole);

A = B;

B = ostatok;

whole = BigInteger.DivRem(A, B, out ostatok);

}

nod = B; //BigInteger.Abs(B);

// Считаем x и y

x = 0;

y = 1;

while (whole\_stack.Count > 0)

{

BigInteger temp = x;

x = y;

y = (temp - y \* whole\_stack.Pop());

}

}

// НОД

private BigInteger NOD\_Euclid(BigInteger A, BigInteger B)

{

if (A == 0 || B == 0)

return 0;

// Чтобы запомнить частные от делений и посчитать потом x и y

Stack<BigInteger> whole\_stack = new Stack<BigInteger>();

// DivRem() - возвращает частное и через out предоставляет остаток от деления

BigInteger whole = BigInteger.DivRem(A, B, out BigInteger ostatok);

while (ostatok != 0)

{

whole\_stack.Push(whole);

A = B;

B = ostatok;

whole = BigInteger.DivRem(A, B, out ostatok);

}

return BigInteger.Abs(B);

}

#endregion

}

}