Изображение выглядит как цепь

Автоматически созданное описание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Автоматизированные системы обработки информации и управления | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Разработка программных систем | | |
| Курс | II | | | Группа | 403 |

Отчёт по лабораторной работе № 2

Вариант № 9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 403 |  |  |  | Шишко Даниил Юрьевич |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |

**Оглавление**

[Постановка задачи 3](#_Toc96790631)

[Исходные данные 3](#_Toc96790632)

[Особые ситуации 3](#_Toc96790633)

[Математические методы и алгоритмы решения задач 3](#_Toc96790634)

[Форматы представления данных 4](#_Toc96790635)

[Структура программы 6](#_Toc96790636)

[Блок-схемы алгоритмов программы 8](#_Toc96790637)

[Описание хода выполнения работы 9](#_Toc96790638)

[Результат работы программы 10](#_Toc96790639)

[Исходный текст программы 11](#_Toc96790640)

[Документирование и комментирование исходного текста 27](#_Toc96790641)

# Постановка задачи

Создать интерфейс ICipher, который определяет методы поддержки шифрования строк. В интерфейсе объявляются два метода Encode() и Decode(), которые используются для шифрования и дешифрования строк, соответственно. Реализовать 2 класса реализующих данный интерфейс. Один из алгоритмов в соответствие с вариантом, второй по выбору.

# Исходные данные

В качестве исходных данных программа использует, вводимое пользователем строку, и ключ, которые шифруются и дешифруются.

# Особые ситуации

Необходимо рассмотреть следующие особые ситуации.

* Пустая строка с ключом или сообщением.
* Определитель матрицы ключа равен нулю.
* НОД определителя матрицы ключа и длинны алфавита равны определителю матрицы.

# Математические методы и алгоритмы решения задач

Шифр Хилла является полиграммным шифром, который может использовать большие блоки с помощью линейной алгебры. Каждой букве алфавита сопоставляется число по модулю 26. Для латинского алфавита часто используется простейшая схема: A = 0, B = 1, …, Z = 25, но это не является существенным свойством шифра. Блок из n букв рассматривается как n-мерный вектор и умножается по модулю 26 на матрицу размера n × n. Если в качестве основания модуля используется число больше чем 26, то можно использовать другую числовую схему для сопоставления буквам чисел и добавить пробелы и знаки пунктуации. Элементы матрицы являются ключом. Матрица должна быть обратима, чтобы была возможность расшифрования.

Для n = 3 система может быть описана так:

Или C =KP (mod 26),

где P и C – векторы-столбцы высоты 3, представляющие открытый и зашифрованный текста соответственно, K – матрица 3x3, представляющая ключ шифрования. Операции выполняются по модулю 26.

Для того, чтобы расшифровать сообщение, требуется получить обратную матрицу ключа . Существуют стандартные методы вычисления обратных матриц (см. способы нахождения обратной матрицы), но не все матрицы имеют обратную (см. обратная матрица). Матрица будет иметь обратную в том и только в том случае, когда её детерминант не равен нулю и не имеет общих делителей с основанием модуля. Если детерминант матрицы равен нулю или имеет общие делители с основанием модуля, то такая матрица не может использоваться в шифре Хилла, и должна быть выбрана другая матрица (в противном случае шифротекст будет невозможно расшифровать). Тем не менее, матрицы, которые удовлетворяют вышеприведенным условиям, существуют в изобилии.

В общем случае, алгоритм шифрования может быть выражен в следующем виде:

Шифрование: C = E (K, P) = KP (mod 26).

Расшифрование: P = D (K, C) = C (mod 26) = KP (mod 26) = P.

Алгоритм Цезаря:

Если сопоставить каждому символу алфавита его порядковый номер (нумеруя с 0), то шифрование и дешифрование можно выразить формулами модульной арифметики:

где x – символ открытого текста, y = символ шифрованного текста, n – мощность алфавита, а k – ключ.

# Форматы представления данных

Программа использует следующие переменные:

Таблица 1 – Переменные, используемые в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| fullAlfabet | string | Алфавит |
| letterQty | int | Длинна алфавита |
| retVal | stirng | Возвращаемое значение |
| plainMessage | string | Сообщение для шифровки |
| key | int | Ключ для шифровки |
| Продолжение таблицы 1 – Переменные, используемые в программе | | |
| encryptedMessage | string | Зашифрованное сообщение |
| message | string | Передаваемое сообщение |
| path | string | Путь к файлу |
| stringFromFile | string | Строка из файла |
| someInput | IInputData | Создатель для типа ввода |
| size | int | Размер |
| isSquare | bool | Квадрат числа |
| matrixOfKey | int[, ] | Матрица ключа |
| asciCode | int | Число в АСКИ2 |
| lenghtOfMessage | int | Длинна сообщения |
| result | int[, ] | Результат вычислений |
| allIntMessage | int[, ] | Матричное сообщение |
| decription | int[, ] | Матрица |
| badData | bool | Неподходящие данные |
| messageFromConsole | string | Строка записанная из консоли |
| randomString | string | Строка сгенерированная случайно |
| letter | int | Символ алфавита |
| number | int | Число |
| text | sting | Строка из консоли |
| choice | int | Выбор пользователя |
| algorithm | ICipher | Алгоритм шифрования |

Программа использует следующие константы:

Таблица 2 – Константы, используемые в программе

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Тип** | **Значение** | **Описание** |
| LowerLimit | const int | 5 | Минимальное число возможных значений |
| UpperLimit | const int | 12 | Максимальное число возможных значений |
| MaxInt | const int | 2147483647 | Значение первого узла в пустом дереве |
| LENGHTOFALPHABET | const int | 37 | Длинна алфавита |

Таблица 3 – Пользовательские типы

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| UserChoice | Способы ввода данных |
| Interaction | Способы взаимодействия с деревом |

В файле каждое сообщения хранятся в одной строке.

# Структура программы

Программа разделена на следующие модули:

1. CaesarCipher.cs – реализация шифра Цезаря;
2. File.cs – ввод/вывод из файла;
3. Input.cs – генерация данных дерева;
4. Program.cs – функция main;
5. HillCipher.cs – реализация шифра Хилла;
6. GetInput.cs – создаёт объекты классов создателей;
7. IInputData.cs – интерфейс для создателей ввода;
8. WorkWithMatrix.cs – операции с матрицой;

Таблица 4 – Функции, составляющие программу

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| CodeEncode | Шифр Цезаря |
| InnerEncode | Преобразования строки в число |
| InnerDecode | Преобразования строки в число |
| Encode | Шифрование строки |
| Decode | Расшифрование строки |
| SaveInFile | Сохранение данных в файл |
| GetDataFromFile | Извлечение данных из файла |
| GetSomeInput | Конструктор для типа ввода |
| GetAlgorithm | Конструктор для алгоритма шифрования |
| GetMatrixFromKey | Преобразование строки в двумерную матрицу |
| GetVectorFromMessage | Преобразование строки в одномерную матрицу |
| GetCount | Возвращает длину сообщения |
| GetNextPartOfMessage | Возвращает следующий блок сообщения |
| Mod | Остаток от деления каждого члена массива |
| SumWithLenghtOfAlphabet | Суммирует каждый член массива с длинной алфавита |
| GetSymbols | Превращает двумерный массив в строку |
| GCD | Нахождение НОД |
| Gcd | Расширенный алгоритм Евклида |
| GetAntiDeterminant | Получение анти детерминанта |
| Encode | Кодирование |
| Decode | Декодирование |
| Продолжение таблицы 4 – Функции, составляющие программу | |
| Input | Ввод данных |
| GetInt | Получить целочисленное значение |
| GetIntKey | Получить целочисленное значение в виде строки |
| GetStringKey | Получить строку |
| GetTransparentMatrix | Транспонировать матрицу |
| GetMatrixOfAlgebraicComplement | Создаёт матрицу алгебраических дополнений |
| GetDeterminant | Вычисляет детерминант |
| GetMinorMatrix | Возвращает минор матрицы |
| Multiplication | Умножение матриц |
| MultiplyOnNumber | Умножение матрицы на число |

# Блок-схемы алгоритмов программы

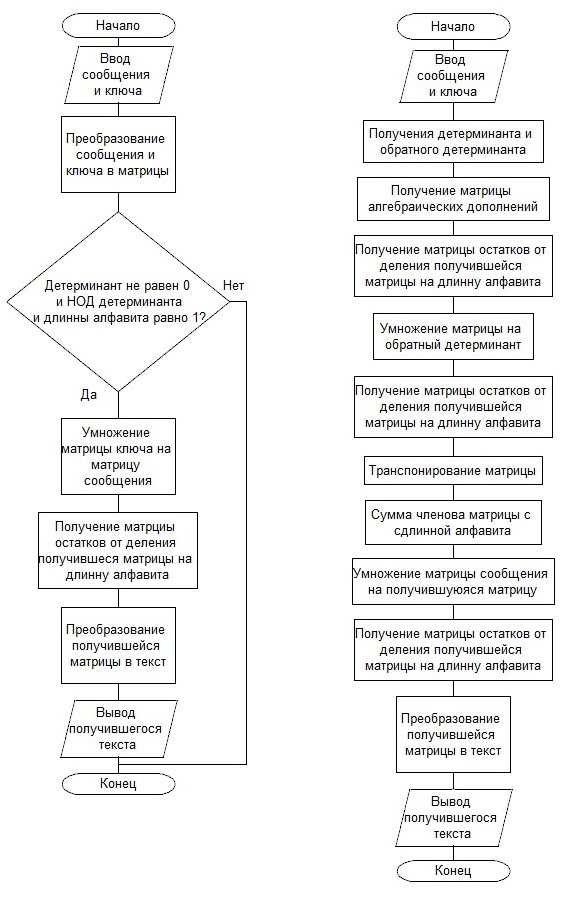


Рисунок 1 – Блок-схема алгоритма решения

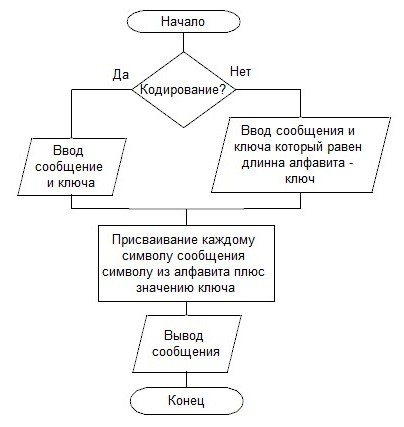


Рисунок 2 – Блок-схема решения алгоритма

# Описание хода выполнения работы

1. В ходе работы было создано решение (Solution) в интегрированной среде разработки Microsoft Visual Studio 2015. В нём был создан проект.
2. Было решено взять усложнённый вариант задания, после двух дней неудачных попыток и из-за нехватки времени на все остальные предметы было решено взять обычное задание.
3. Созданы исключения, которые нужны для алгоритма Хилла, тем самым исправлены возникающие ошибки при преобразовании.

# Результат работы программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Результат работы программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – Результат работы программы

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 5 – Результат работы программы

# Исходный текст программы

[ Начало программы ---]

[CaesarCipher.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace HillCipher

{

public class CaesarCipher : ICipher

{

const string alfabet = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ., ?";

private static string CodeEncode(string text, int k)

{

var fullAlfabet = alfabet + alfabet.ToLower();

var letterQty = fullAlfabet.Length;

var retVal = "";

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

{

var c = text[i];

var index = fullAlfabet.IndexOf(c);

if (index < 0)

{

retVal += c.ToString();

}

else

{

var codeIndex = (letterQty + index + k) % letterQty;

retVal += fullAlfabet[codeIndex];

}

}

return retVal;

}

private static string InnerEncode(string plainMessage, int key)

=> CodeEncode(plainMessage, key);

private static string InnerDecode(string encryptedMessage, int key)

=> CodeEncode(encryptedMessage, -key);

public string Encode(string message, string key)

{

return InnerEncode(message, Int32.Parse(key));

}

public string Decode(string message, string key)

{

return InnerDecode(message, Int32.Parse(key));

}

}

}

[CaesarCipher.cs]

[File.cs]

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

namespace HillCipher

{

class File

{

static public void SaveInFile(string message)

{

string path = "";

do

{

Console.WriteLine("Please enter path to file");

path = Console.ReadLine();

FileInfo tempFile = new FileInfo(path);

if (!tempFile.Exists)

{

try

{

StreamWriter newFile = new StreamWriter(path);

newFile.Close();

}

catch

{

Console.WriteLine("Bad name for file, please try again");

continue;

}

}

if (tempFile.Exists)

{

if (tempFile.IsReadOnly)

{

Console.WriteLine("Something wrong with file. please try again");

continue;

}

else

{

Console.WriteLine("Do you want to rewrite file?" + Environment.NewLine + "1”+

“– Yes");

if (SomeInput.GetInt() != 1)

{

continue;

}

}

}

tempFile.Delete();

break;

}

while (true);

StreamWriter file = new StreamWriter(path);

file.WriteLine(message);

file.Close();

Console.WriteLine("File saved");

}

static public string GetDataFromFile()

{

string stringFromFile = "";

string path = "";

do

{

Console.WriteLine("Please enter path to file");

path = Console.ReadLine();

FileInfo tempFile = new FileInfo(path);

if (!tempFile.Exists)

{

Console.WriteLine("File is not existing");

continue;

}

StreamReader tempOpenedFile = new StreamReader(path);

string tempString = "";

while (!tempOpenedFile.EndOfStream)

{

tempString += tempOpenedFile.ReadLine();

}

tempOpenedFile.Close();

bool badData = false;

foreach (char letter in tempString)

{

if ((letter <= 'А' && letter >= 'Я') || letter != ',' || letter != '.' ||

letter != ' ' || letter != '?' || letter != 'Ё')

Console.WriteLine("Bad data");

badData = true;

break;

}

if (badData)

{

continue;

}

break;

}

while (true);

StreamReader file = new StreamReader(path, false);

while (!file.EndOfStream)

{

stringFromFile += file.ReadLine();

}

file.Close();

return stringFromFile;

}

}

}

[File.cs]

[GetInput.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace HillCipher

{

static class GetInput

{

public enum UserChoice

{

ConsoleInput = 1,

FileInput,

RandomInput,

End

}

static public IInputData GetSomeInput(int choice)

{

IInputData someInput = null;

if((UserChoice)choice == UserChoice.ConsoleInput)

{

someInput = new ManualInput();

}

if ((UserChoice)choice == UserChoice.RandomInput)

{

someInput = new RandomInput();

}

if ((UserChoice)choice == UserChoice.FileInput)

{

someInput = new FileInput();

}

return someInput;

}

static public ICipher GetAlgorithm(int choice)

{

if((ICipher.Algorithms)choice == ICipher.Algorithms.CaesarCipher)

{

return new CaesarCipher();

}

if ((ICipher.Algorithms)choice == ICipher.Algorithms.HillCipher)

{

return new HillCipher();

}

return null;

}

}

}

[GetInput.cs]

[HillCipher.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace HillCipher

{

public class HillCipher : ICipher

{

public const int LENGHTOFALPHABET = 37;

private static int[,] GetMatrixFromKey(string key)

{

int size = key.Length;

double result = Math.Sqrt(size);

bool isSquare = result % 1 == 0;

if (!isSquare)

{

while (!isSquare)

{

size++;

result = Math.Sqrt(size);

isSquare = result % 1 == 0;

}

}

int n = (int)Math.Sqrt(size);

int[,] matrixOfKey = new int[n, n];

int count = 0;

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

{

for (int j = 0; j < n; j++)

{

if (i \* n + j < key.Length)

{

int asciCode = (int)key[count];

if (asciCode >= 'А' && asciCode <= 'Д')

{

matrixOfKey[i, j] = asciCode - 'А';

}

if (asciCode == 'Ё')

{

matrixOfKey[i, j] = 6;

}

if (asciCode > 'Д' && asciCode <= 'Я')

{

matrixOfKey[i, j] = asciCode - 'А' + 1;

}

if (asciCode == '.')

{

matrixOfKey[i, j] = 33;

}

if (asciCode == ',')

{

matrixOfKey[i, j] = 34;

}

if (asciCode == ' ')

{

matrixOfKey[i, j] = 35;

}

if (asciCode == '?')

{

matrixOfKey[i, j] = 36;

}

count++;

}

else

{

matrixOfKey[i, j] = 35;

}

}

}

return matrixOfKey;

}

private static int[,] GetVectorFromMessage(string message, int size)

{

int lenghtOfMessage = message.Length;

if (lenghtOfMessage % size != 0)

{

lenghtOfMessage += (size - (lenghtOfMessage % size));

}

int[,] vectorOfMessage = new int[lenghtOfMessage / size, size];

for (int j = 0; j < lenghtOfMessage / size; j++)

{

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

{

if (j \* size + i < message.Length)

{

int asciCode = (int)message[j \* size + i];

if (asciCode >= 'А' && asciCode <= 'Д')

{

vectorOfMessage[j, i] = asciCode - 'А';

}

if (asciCode == 'Ё')

{

vectorOfMessage[j, i] = 6;

}

if (asciCode > 'Д' && asciCode <= 'Я')

{

vectorOfMessage[j, i] = asciCode - 'А' + 1;

}

if (asciCode == '.')

{

vectorOfMessage[j, i] = 33;

}

if (asciCode == ',')

{

vectorOfMessage[j, i] = 34;

}

if (asciCode == ' ')

{

vectorOfMessage[j, i] = 35;

}

if (asciCode == '?')

{

vectorOfMessage[j, i] = 36;

}

}

else

{

vectorOfMessage[j, i] = 35;

}

}

}

return vectorOfMessage;

}

private static int GetCount(int sizeOfMessage, int sizeOfOneDemention)

{

int lenghtOfMessage = sizeOfMessage;

if (lenghtOfMessage % sizeOfOneDemention != 0)

{

lenghtOfMessage += (sizeOfOneDemention - (lenghtOfMessage % sizeOfOneDemention));

}

lenghtOfMessage = lenghtOfMessage / sizeOfOneDemention;

return lenghtOfMessage;

}

private static int[,] GetNextPartOfMessage(int[,] matrix, int iteration)

{

int[,] result = new int[1, matrix.GetLength(1)];

for (int i = 0; i < matrix.GetLength(1); i++)

{

result[0, i] = matrix[iteration, i];

}

return result;

}

private static int[,] Mod(int[,] array)

{

int[,] result = new int[array.GetLength(0), array.GetLength(1)];

for (int i = 0; i < array.GetLength(0); i++)

{

for (int j = 0; j < array.GetLength(1); j++)

{

result[i, j] = array[i, j] % LENGHTOFALPHABET;

}

}

return result;

}

private static int[,] SumWithLenghtOfAlphabet(int[,] matrix)

{

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

if (matrix[i, j] < 0)

{

matrix[i, j] = LENGHTOFALPHABET + matrix[i, j];

}

}

}

return matrix;

}

private static string GetSymbols(int[,] array)

{

string result = "";

for (int i = 0; i < array.GetLength(1); i++)

{

if (array[0, i] >= 0 && array[0, i] < 6)

{

result += (char)(array[0, i] + 'А');

}

if (array[0, i] == 6)

{

result += 'Ё';

}

if (array[0, i] > 6 && array[0, i] <= 32)

{

result += (char)(array[0, i] - 1 + 'А');

}

if (array[0, i] == 33)

{

result += '.';

}

if (array[0, i] == 34)

{

result += ',';

}

if (array[0, i] == 35)

{

result += ' ';

}

if (array[0, i] == 36)

{

result += '?';

}

}

return result;

}

private static int GCD(int a, int b)

{

while (b != 0)

{

var temp = b;

b = a % b;

a = temp;

}

return a;

}

private static (int x, int y, int a) Gcd(int a, int b)

{

if (b == 0)

{

return (1, 0, a);

}

var (y, x, g) = Gcd(b, a % b);

return (x, y - ((int)(a / b) \* x), g);

}

private static int GetAntiDeterminant(int det, int x)

{

int toReturn = 0;

if (det < 0 && x >= 0)

{

toReturn = x;

}

if (det >= 0 && x < 0)

{

toReturn = LENGHTOFALPHABET + x;

}

if (det >= 0 && x >= 0)

{

toReturn = x;

}

if (det < 0 && x < 0)

{

toReturn = -x;

}

return toReturn;

}

public string Encode(string message, string key)

{

string encriptedString = "";

var intKey = GetMatrixFromKey(key);

int det = WorkWithMatrix.GetDeterminant(intKey);

if (WorkWithMatrix.GetDeterminant(intKey) == 0) throw new DivideByZeroException();

if (GCD(WorkWithMatrix.GetDeterminant(intKey), LENGHTOFALPHABET) != 1) throw new

DivideByZeroException();

var (x, y, z) = Gcd(det, LENGHTOFALPHABET);

x = GetAntiDeterminant(det, x);

if (x == 0) throw new DivideByZeroException();

var allIntMessage = GetVectorFromMessage(message, intKey.GetLength(0));

for (int i = 0; i < GetCount(message.Length, intKey.GetLength(0)); i++)

{

var intMessage = GetNextPartOfMessage(allIntMessage, i);

var result = WorkWithMatrix.Multiplication(intMessage, intKey);

result = Mod(result);

encriptedString += GetSymbols(result);

}

return encriptedString;

}

public string Decode(string message, string key)

{

string decryptedString = "";

var intKey = GetMatrixFromKey(key);

if (WorkWithMatrix.GetDeterminant(intKey) == 0) throw new DivideByZeroException();

if (GCD(WorkWithMatrix.GetDeterminant(intKey), LENGHTOFALPHABET) ==

WorkWithMatrix.GetDeterminant(intKey)) throw new DivideByZeroException();

var allIntMessage = GetVectorFromMessage(message, intKey.GetLength(0));

for (int i = 0; i < GetCount(message.Length, intKey.GetLength(0)); i++)

{

var intMessage = GetNextPartOfMessage(allIntMessage, i);

int det = WorkWithMatrix.GetDeterminant(intKey);

var (x, y, z) = Gcd(det, LENGHTOFALPHABET);

x = GetAntiDeterminant(det, x);

var decription = WorkWithMatrix.GetMatrixOfAlgebraicComplement(intKey);

decription = Mod(decription);

decription = WorkWithMatrix.MultiplyOnNumber(decription, x);

decription = Mod(decription);

decription = WorkWithMatrix.GetTransparentMatrix(decription);

decription = SumWithLenghtOfAlphabet(decription);

decription = WorkWithMatrix.Multiplication(intMessage, decription);

decription = Mod(decription);

decryptedString += GetSymbols(decription);

}

return decryptedString.Trim();

}

}

}

[HillCipher.cs]

[ICipher.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace HillCipher

{

interface ICipher

{

public enum Algorithms

{

HillCipher = 1,

CaesarCipher

}

abstract string Encode(string message, string key);

abstract string Decode(string message, string key);

}

}

[ICipher.cs]

[IInputData.cs]

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

namespace HillCipher

{

public interface IInputData

{

string Input();

}

public class ManualInput : IInputData

{

public string Input()

{

string messageFromConsole;

bool badData;

do

{

badData = false;

Console.WriteLine("Please enter message to encypt");

string tempString = Console.ReadLine();

foreach (char letter in tempString)

{

if (!((letter >= 'А' && letter <= 'Я') || (letter == ',' || letter == '.' ||

letter == ' ' || letter == '?' || letter == 'Ё')))

{

Console.WriteLine("Bad data");

badData = true;

break;

}

}

if (badData)

continue;

messageFromConsole = tempString;

break;

}

while (true);

Console.WriteLine("Do you want to save data in file?" + Environment.NewLine + "1”+

“ - yes");

if (SomeInput.GetInt() == 1)

{

File.SaveInFile(messageFromConsole);

}

return messageFromConsole;

}

}

public class RandomInput : IInputData

{

const int LowerLimit = 5;

const int UpperLimit = 12;

static Random rnd = new Random();

public string Input()

{

string randomString = "";

int countOfNLetter = rnd.Next(LowerLimit, UpperLimit);

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

{

var letter = rnd.Next(HillCipher.LENGHTOFALPHABET - 1);

if (letter >= 0 && letter < 6)

{

randomString += (char)(letter + 'А');

}

if (letter == 6)

{

randomString += 'Ё';

}

if (letter > 6 && letter <= 32)

{

randomString += (char)(letter - 1 + 'А');

}

if (letter == 33)

{

randomString += '.';

}

if (letter == 34)

{

randomString += ',';

}

if (letter == 35)

{

randomString += ' ';

}

if (letter == 36)

{

randomString += '?';

}

}

Console.WriteLine(randomString);

Console.WriteLine("Do you want to save data in file?" + Environment.NewLine + "1”+

“ - yes");

if (SomeInput.GetInt() == 1)

{

File.SaveInFile(randomString);

}

return randomString;

}

}

public class FileInput : IInputData

{

public string Input()

{

string stringFromFile = "";

string path;

do

{

Console.WriteLine("Please enter path to file");

path = Console.ReadLine();

FileInfo tempFile = new FileInfo(path);

if (!tempFile.Exists)

{

Console.WriteLine("File is not existing");

continue;

}

StreamReader tempOpenedFile = new StreamReader(path);

string tempString = "";

while (!tempOpenedFile.EndOfStream)

{

tempString += tempOpenedFile.ReadLine();

}

tempOpenedFile.Close();

bool badData = false;

foreach(char letter in tempString)

{

if (!((letter >= 'А' && letter <= 'Я') || (letter == ',' || letter == '.' ||

letter == ' ' || letter == '?' || letter == 'Ё')))

{

Console.WriteLine("Bad data");

badData = true;

break;

}

}

if (badData)

{

continue;

}

break;

}

while (true);

StreamReader file = new StreamReader(path, false);

while (!file.EndOfStream)

{

stringFromFile += file.ReadLine();

}

file.Close();

Console.WriteLine(stringFromFile);

return stringFromFile;

}

}

}

[IInputData.cs]

[Input.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace HillCipher

{

class SomeInput

{

static public int GetInt()

{

int number = 0;

while (true)

{

Console.Write("Please enter integer: ");

string text = Console.ReadLine();

if (int.TryParse(text, out number))

{

break;

}

Console.WriteLine("This is not a integer, try again");

}

return number;

}

static public string GetIntKey()

{

string text = "";

while (true)

{

Console.Write("Please enter integer: ");

text = Console.ReadLine();

if (int.TryParse(text, out int number))

{

break;

}

Console.WriteLine("This is not a integer, try again");

}

return text;

}

static public string GetStringKey()

{

string key;

bool badData;

do

{

badData = false;

Console.WriteLine("Please enter key");

string tempString = Console.ReadLine();

foreach (char letter in tempString)

{

if (!((letter >= 'А' && letter <= 'Я') || (letter == ',' || letter == '.' ||

letter == ' ' || letter == '?' || letter == 'Ё')))

{

Console.WriteLine("Bad data");

badData = true;

break;

}

}

if (badData)

continue;

key = tempString;

break;

}

while (true);

return key;

}

}

}

[Input.cs]

[Program.cs]

using System;

using System.Text;

namespace HillCipher

{

class Program

{

static void Main(string[] args)

{

string key, message;

int count, choice;

IInputData someInput;

ICipher algorithm;

Console.WriteLine("Created by: Shishko Daniil Yrevich" + Environment.NewLine +

"Option: 9" + Environment.NewLine + "Aim: Create an ICipher interface that”+

“defines methods to support string encryption. " +

"The interface declares two methods Encode() and Decode(), which are used to”+

“encrypt and decrypt strings" +

", respectively. Implement 2 classes implementing this interface." +

Environment.NewLine +

"Welcome.");

do

{

count = 1;

Console.WriteLine(Environment.NewLine + Environment.NewLine + "1 - Console Input"

+ Environment.NewLine

+ "2 - File Input" + Environment.NewLine + "3 - Random Input" +

Environment.NewLine + "4 - End program");

choice = SomeInput.GetInt();

if ((GetInput.UserChoice)choice < GetInput.UserChoice.ConsoleInput ||

(GetInput.UserChoice)choice > GetInput.UserChoice.End)

{

Console.WriteLine("We don't have these choice, please try again");

continue;

}

if((GetInput.UserChoice)choice == GetInput.UserChoice.End)

{

break;

}

someInput = GetInput.GetSomeInput(choice);

message = someInput.Input();

do

{

Console.WriteLine();

Console.WriteLine("Please choose the algorithm");

foreach (string name in Enum.GetNames(typeof(ICipher.Algorithms)))

{

Console.WriteLine(count.ToString() + " - " + name);

count++;

}

choice = SomeInput.GetInt();

if ((ICipher.Algorithms)choice < ICipher.Algorithms.HillCipher ||

(ICipher.Algorithms)choice > ICipher.Algorithms.CaesarCipher)

{

Console.WriteLine("We don't have these choice, please try again");

continue;

}

algorithm = GetInput.GetAlgorithm(choice);

Console.WriteLine();

if ((ICipher.Algorithms)choice == ICipher.Algorithms.HillCipher)

{

key = SomeInput.GetStringKey();

break;

}

if ((ICipher.Algorithms)choice == ICipher.Algorithms.CaesarCipher)

{

key = SomeInput.GetIntKey();

break;

}

} while (true);

do

{

Console.WriteLine(Environment.NewLine + "1 - To Encode" + Environment.NewLine +

"2 - To Decode");

choice = SomeInput.GetInt();

if(choice == 1)

{

message = algorithm.Encode(message, key);

Console.WriteLine("Encoded string: " + message);

Console.WriteLine("Do you want to save data in file?" + Environment.NewLine +

"1 - yes");

if(SomeInput.GetInt() == 1)

{

File.SaveInFile(message);

}

}

if (choice == 2)

{

message = algorithm.Decode(message, key);

Console.WriteLine("Decoded string: " + message);

Console.WriteLine("Do you want to save data in file?" + Environment.NewLine +

"1 - yes");

if (SomeInput.GetInt() == 1)

{

File.SaveInFile(message);

}

}

else

{

Console.WriteLine("We don't have these choice, please try again");

continue;

}

Console.WriteLine(Environment.NewLine + "Enter 0 to return");

if (SomeInput.GetInt() == 0)

break;

} while (true);

}

while (true);

}

}

}

[Program.cs]

[WorkWothMatrix.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace HillCipher

{

class WorkWithMatrix

{

public static int[,] GetTransparentMatrix(int[,] matrix)

{

int count = 0;

for (int i = 1; i < matrix.GetLength(0); i++)

{

count++;

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

{

int temp = matrix[i, j];

matrix[i, j] = matrix[j, i];

matrix[j, i] = temp;

}

}

return matrix;

}

public static int[,] GetMatrixOfAlgebraicComplement(int[,] matrix)

{

int[,] result = new int[matrix.GetLength(0), matrix.GetLength(0)];

int sign = 1;

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(0); j++)

{

sign = ((i + 1) % 2 == (j + 1) % 2) ? 1 : -1;

int det = GetDeterminant(GetMinorMatrix(matrix, i, j));

result[i, j] = sign \* det;

}

}

return result;

}

public static int GetDeterminant(int[,] matrix)

{

if (matrix.GetLength(0) == 0 && matrix.GetLength(1) == 0)

return 1;

if (matrix.GetLength(0) != matrix.GetLength(1))

throw new Exception("Matrix should be square!");

if (matrix.GetLength(0) == 2)

return matrix[0, 0] \* matrix[1, 1] - matrix[0, 1] \* matrix[1, 0];

int sign = 1;

int result = 0;

int j = 0;

for (int i = 0; i < matrix.GetLength(0); i++)

{

sign = ((i + 1) % 2 == (j + 1) % 2) ? 1 : -1;

result += sign \* matrix[i, j] \* GetDeterminant(GetMinorMatrix(matrix, i, j));

}

return result;

}

static public int[,] GetMinorMatrix(int[,] matrix, int row, int col)

{

int[,] result = new int[matrix.GetLength(0) - 1, matrix.GetLength(1) - 1];

int m = 0, k;

for (int i = 0; i < matrix.GetLength(0); i++)

{

if (i == row) continue;

k = 0;

for (int j = 0; j < matrix.GetLength(1); j++)

{

if (j == col) continue;

result[m, k++] = matrix[i, j];

}

m++;

}

return result;

}

static public int[,] Multiplication(int[,] a, int[,] b)

{

if (a.GetLength(1) != b.GetLength(0)) throw new Exception("Mtrix can’t be”+

“multiply");

int[,] r = new int[a.GetLength(0), b.GetLength(1)];

for (int i = 0; i < a.GetLength(0); i++)

{

for (int j = 0; j < b.GetLength(1); j++)

{

for (int k = 0; k < b.GetLength(0); k++)

{

r[i, j] += a[i, k] \* b[k, j];

}

}

}

return r;

}

public static int[,] MultiplyOnNumber(int[,] array, double number)

{

int[,] result = new int[array.GetLength(0), array.GetLength(1)];

for (int i = 0; i < array.GetLength(0); i++)

{

for (int j = 0; j < array.GetLength(1); j++)

{

result[i, j] = (int)(array[i, j] \* number);

}

}

return result;

}

}

}

[WorkWothMatrix.cs]

[ --- Конец программы]

# Документирование и комментирование исходного текста