|  |  |  |  |
| --- | --- | --- | --- |
| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка  ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  Кафедра програмних систем і технологій  **Практична робота №1**  з дисципліни  **«Захист програм та даних»**  **Тема :** **«Шифри заміни»** | | | |
| Виконала: | Ємець  Євгенія  Ярославівна |  | |
| Група | ІПЗ-33 |  |  |
|  |  |  |  |
|  |  |
| Перевірила: | Супрун  Ольга  Миколаївна |
| Дата перевірки |  |
| Київ 2018 | | | |

**Хід роботи**

**Завдання №1**

Необхідно зашифрувати своє прізвище за допомогою шифру Цезаря

Шифр Цезаря — симетричний алгоритм шифрування підстановками. Принцип дії полягає в тому, щоб циклічно зсунути алфавіт, а ключ — це кількість літер, на які робиться зсув.

**using** System;  
**namespace** Caesar  
{  
 **static class** Program  
 {  
 **private static void** Main(**string**[] args)  
 {  
 **const int *n*** = 1;  
   
 Console.WriteLine(**"Введите слово,которое нужно зашифровать:"**);  
 **var** s = Console.ReadLine();*//храню слово, которое будем шифровать* Console.WriteLine(**"Введите ключ:"**);  
 **var** key = Convert.ToInt32(Console.ReadLine());  
   
 **var** s1 = **""**;*//храню результат шифрования* **const string *alfphabet*** = **"абвгґдеєжзиіїйклмнопрстуфхцчщьюя"**; *//украинский алфавит* **var** m = ***alfphabet***.**Length**; *//количество знаков в алфавите* **if** (s != **null**)  
 {  
 **foreach** (**var** t **in** s)  
 {  
 **for** (**var** j = 0; j < ***alfphabet***.**Length**; j++) *//цикл сравнения каждой бкувы с алфавитом* {  
 **if** (t != ***alfphabet***[j]) **continue**;  
   
 **var** temp = j \* ***n*** + key; *//номер буквы+сдвиг в темп* **while** (temp >= m) *//чтобы темп не уходил за рамки алфавита* temp -= m;  
  
 s1 = s1 + ***alfphabet***[temp]; *//заношу зашифрованную букву в переменную для ее хранения* }  
 }  
 Console.WriteLine(**"Зашифрованное слово:"** + s1);  
 }  
 **else** Console.WriteLine(**"String is null"**);  
 Console.ReadKey(**true**);  
 }  
 }  
}

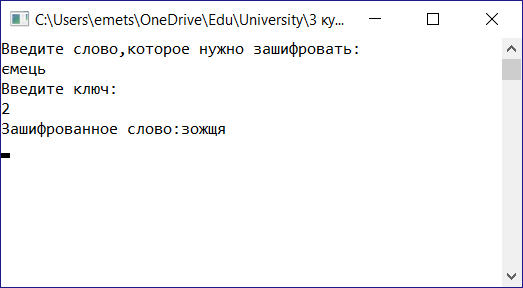


Рис. 1 – Результат роботи шифру Цезаря

**Завдання №2**

Необхідно зашифрувати своє прізвище за допомогою лозунгового шифру

Для даного шифру побудова таблиці шіфрозамін засноване на гаслі (ключі) - легко запам'ятовується слові. Другий рядок таблиці шіфрозамін заповнюється спочатку словом-гаслом (причому повторювані букви відкидаються), а потім іншими літерами, що не ввійшли в слово-гасло, в алфавітному порядку.

**using** System;  
**using** System.Linq;  
**using** System.Text;  
  
**namespace** Лозунговый\_шифр  
{  
 **internal class** SloganCipher  
 {  
 **private char**[] **\_table**;  
 **private** SloganCipher(**string** slogan)  
 {  
 **Slogan** = slogan ?? **throw new** ArgumentNullException(**nameof**(slogan));  
 CreateTable();  
 }  
 **private string Slogan** {  
 **get**;  
 }  
 **private string** Cipher(**string** text)  
 {  
 **if** (**string**.IsNullOrEmpty(text)) **return string**.**Empty**;  
 **var** buffer = **new** StringBuilder(text.**Length**);  
 **foreach** (**var** letter **in** text)  
 buffer.Append(IsRussianLetter(letter) ? **\_table**[letter - **'а'**] : letter);  
 **return** buffer.ToString();  
 }  
 **private void** CreateTable()  
 {  
 **\_table** = **Slogan**.Where(IsRussianLetter).Distinct().Concat(**"абвгдежзиклмнопрстуфхцчшщыьъэюя"**.Except(**Slogan**)).ToArray();  
 }  
 **private static bool** IsRussianLetter(**char** character)  
 {  
 **return char**.IsLetter(character) && character >= **'а'** && character <= **'я'**;  
 }  
   
 **static void** Main()  
 {  
 Console.Write(**"Лозунг: "**);  
 **var** slogan = Console.ReadLine();  
   
 **var** cipher = **new** SloganCipher(slogan);  
   
 Console.Write(**"Текст для шифрования: "**);  
 **var** text = Console.ReadLine();  
   
 **var** cipheredText = cipher.Cipher(text);  
 Console.WriteLine(**"Зашифрованный текст: {0}"**, cipheredText);  
  
 Console.ReadKey(**true**);  
 }  
 }  
}

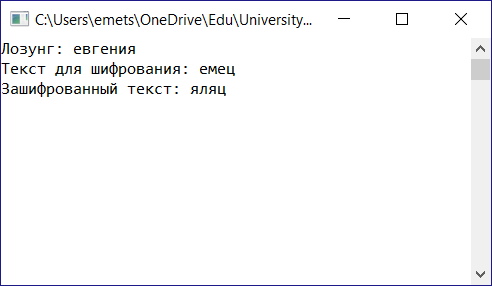


Рис. 2 – лозунговий шифр

**Завдання №3**

Необхідно зашифрувати своє прізвище за допомогою шифру квадрат Полібія

У криптографії квадрат Полібія, також відомий як шахова дошка Полібія — оригінальний код простої заміни, одна з найдавніших систем кодування, запропонована Полібієм (грецький історик, полководець, державний діяч, III століття до н. е.). Цей спосіб кодування спочатку застосовувався для грецької абетки, але потім поширився на інші мови.

**using** System;  
  
**namespace** Polybius\_Square{  
 **internal static class** Program{  
 **private static void** Main(){  
 **const string *randomLine*** = **"ЙЦУКЕНГШЩЗХФЪЫВАПРОЛДЖЭЧСМИТЬБЮ"**;  
 **var** rand = **new** Random();  
   
 Console.WriteLine(**"Введите сообщение:"**);  
 **var** line = Console.ReadLine()?.ToUpper().Replace(**" "**, **""**);  
   
 **if** (line != **null**)  
 {  
 **var** d = (**int**)Math.Ceiling(Math.Sqrt(line.**Length**));  
 **if** (d % 2 != 1)  
 d++;  
 Console.WriteLine(**"Квадрат: "** + d.ToString());  
   
 **var** quad = **new int**[d, d];  
 **for** (**var** j = 0; j < d; j++){   
 **for** (**var** i = 0; i < d; i++){  
 quad[i, j] = d\*(((i + 1) + (j + 1) - 1 + (d / 2)) % d)+(((i+1) + 2\*(j+1) - 2) % d) + 1;  
 Console.Write(quad[i, j].ToString() + **"\t"**); }  
 Console.WriteLine(); }  
   
 Console.WriteLine(**"Шифрование сообщения:"**);  
 **var** cryptedString = **""**;  
 **for** (**var** j = 0; j < d; j++){   
 **for** (**var** i = 0; i < d; i++){   
 **if** ((quad[i, j] - 1) < line.**Length**){  
 Console.Write(line[quad[i, j] - 1] + **"\t"**);  
 cryptedString += line[quad[i, j] - 1]; }  
 **else**{  
 **var** randomChar = ***randomLine***[rand.Next(0, ***randomLine***.**Length** - 1)];  
 Console.Write(randomChar + **"\t"**);  
 cryptedString += randomChar; } }  
 Console.WriteLine(); }  
 Console.WriteLine(**"\nЗашифрованная строка:"**);  
 Console.WriteLine(cryptedString); }  
 Console.ReadKey(); } } }

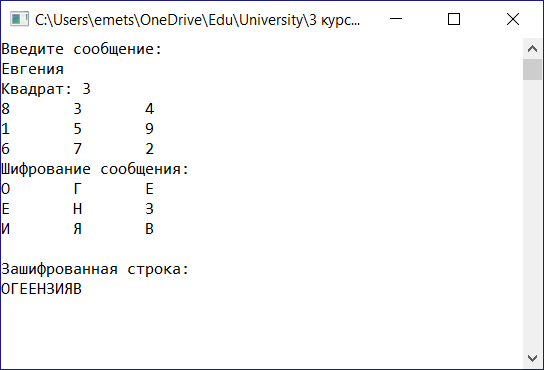


Рис. 3 – шифр Полібія

**Завдання №4**

Необхідно зашифрувати своє прізвище за допомогою шифруючої системи Трисемуса

Для отримання такого шифру заміни зазвичай використовувалися таблиця для запису букв алфавіту і ключове слово (або фраза). У таблицю спочатку вписувалося за термінами ключове слово, причому повторювані букви відкидалися. Потім ця таблиця доповнювалася не увійшли в неї літерами алфавіту по порядку. Зображена таблиця з ключовим словом «Дядина».

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Д | Я | И | Н | А | Б |
| В | Г | Е | Ё | Ж | З |
| Й | К | Л | М | О | П |
| Р | С | Т | У | Ф | Х |
| Ц | Ч | Ш | Щ | Ы | Ь |
| Ъ | Э | Ю | - | - | - |

Таблиця шіфрозамін для шифру Трисемуса

Кожна буква відкритого повідомлення замінюється буквою, розташованої під нею в тому ж стовпці. Якщо буква знаходиться в останньому рядку таблиці, то для її шифрування беруть саму верхню букву стовпця.

**using** System;  
**using** System.Linq;  
  
**namespace** TrisemusSystem  
{  
 **internal class** Program  
 {  
 **public static void** Main(**string**[] args)  
 {  
 **var** alphabet = **"АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ ,."**.ToCharArray();  
   
 *// Пытаемся вычислить размерность таблицы* Console.WriteLine(**"Символов в алфавите: "** + alphabet.**Length**);  
 **int** rows = 0, columns;  
 **bool** isValidTable;  
   
 **do** {  
 Console.Write(**"Количество колонок в таблице: "**);  
 isValidTable = **int**.TryParse(Console.ReadLine(), **out** columns) && columns > 1;  
 **if** (!isValidTable)  
 {  
 Console.WriteLine(**"Необходимо ввести число больше 1"**);  
 }  
 **else** {  
 rows = alphabet.**Length** / columns;  
 isValidTable &= rows > 1 && rows \* columns == alphabet.**Length**;  
 **if** (!isValidTable)  
 {  
 Console.WriteLine(**"Необходимо ввести число колонок таким образом, чтобы число строк таблицы "** +  
 **"было больше 1 и таблица могла вмещать в себе все символы алфавита"**);  
 }  
 }  
 }  
 **while** (!isValidTable);  
   
 *// Пытаемся получить ключевое слово* **char**[] keyWord;  
 **bool** isValidKeyWord;  
 **do** {  
 Console.Write(**"Введите ключевое слово: "**);  
 keyWord = Console.ReadLine()?.ToUpper().Distinct().ToArray();  
 isValidKeyWord = keyWord.**Length** > 0 && keyWord.**Length** <= alphabet.**Length**;  
 **if** (!isValidKeyWord)  
 {  
 Console.WriteLine(**"Ключевое слово не может быть пустой строкой или содержать число уникальных символов больше размера алфавита"**);  
 }  
 **else** {  
 isValidKeyWord &= !keyWord.Except(alphabet).Any();  
 **if** (!isValidKeyWord)  
 {  
 Console.WriteLine(**"Ключевое слово не может содержать символы, которых нет в алфавите"**);  
 }  
 }  
 }  
 **while** (!isValidKeyWord);  
   
 *// Создаем таблицу* **var** table = **new char**[rows, columns];  
   
 *// Вписываем в нее ключевое слово* **for** (**var** i = 0; i < keyWord.**Length**; i++)  
 {  
 table[i / columns, i % columns] = keyWord[i];  
 }  
   
 *// Исключаем уникальные символы ключевого слова из алфавита* alphabet = alphabet.Except(keyWord).ToArray();  
   
 *// Вписываем алфавит* **for** (**var** i = 0; i < alphabet.**Length**; i++)  
 {  
 **var** position = i + keyWord.**Length**;  
 table[position / columns, position % columns] = alphabet[i];  
 }  
   
 *// Получаем сообщение, которое необходимо зашифровать* **string** message;  
 **bool** isValidMessage;  
 **do** {  
 Console.Write(**"Введите сообщение: "**);  
 message = Console.ReadLine()?.ToUpper();  
 isValidMessage = !**string**.IsNullOrEmpty(message);  
 **if** (!isValidMessage)  
 {  
 Console.WriteLine(**"Сообщение не может быть пустой строкой"**);  
 }  
 }  
 **while** (!isValidMessage);  
   
 *// Создаем место для будущего зашифрованного сообщения* **var** result = **new char**[message.**Length**];  
   
 *// Шифруем сообщение* **for** (**var** k = 0; k < message.**Length**; k++)  
 {  
 **var** symbol = message[k];  
 *// Пытаемся найти символ в таблице* **for** (**var** i = 0; i < rows; i++)  
 {  
 **for** (**var** j = 0; j < columns; j++)  
 {  
 **if** (symbol != table[i, j]) **continue**;  
 symbol = table[(i + 1) % rows, j]; *// Смещаемся циклически на следующую строку таблицы и запоминаем новый символ* i = rows; *// Завершаем цикл по строкам* **break**; *// Завершаем цикл по колонкам* }  
 }  
 *// Записываем найденный символ* result[k] = symbol;  
 }  
   
 *// Выводим зашифрованное сообщение* Console.WriteLine(**"Зашифрованное сообщение: "** + **new string**(result));

Console.ReadKey(**true**);  
 }  
 }  
}

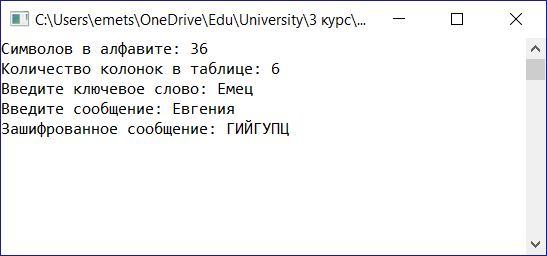


Рис. 4 - шифр Трисемуса

**Завдання №5**

Необхідно зашифрувати своє прізвище за допомогою шифра Playfair

Шифр Плейфера — ручна симетрична техніка шифрування, в якій вперше використано заміну біграмм. Винайдено в 1854 році англійським фізиком Чарльзом Вітстоном, але названа ім'ям лорда Лайона Плейфера, який зробив великий внесок в просування використання даної системи шифрування в державній службі. Шифр передбачає шифрування пар символів (біграм) замість одиночних символів, як в шифрі підстановки й у більш складних системах шифрування Віженера. Таким чином, шифр Плейфера більш стійкий до злому в порівнянні з шифром простої заміни, так як ускладнюється його частотний аналіз. Він може бути проведений, але не для 26 можливих символів (латинський алфавіт), а для 26 х 26 = 676 можливих біграм, і значно більш трудомісткий та вимагає більшого обсягу зашифрованого тексту.

**using** System;  
**using** System.Collections.Generic;  
**using** System.Text;  
  
**namespace** Playfair  
{  
 **public abstract class** PlayFairSettings  
 {  
 **private** HashSet<**char**> **HsAlphabet** { **get**; set; }  
 **protected** Dictionary<**char**, **char**> **Replaces** { **get**; **private** set; }  
   
 **protected abstract** IEnumerable<**char**> **AlphabetChars** { **get**; }  
   
 **public abstract int Columns** { **get**; }  
 **public abstract int Rows** { **get**; }  
   
 **public abstract char Replacer** { **get**; }  
   
 **public** IEnumerable<**char**> **Alphabet** {  
 **get** {  
 **var** chars = **AlphabetChars**;  
 **foreach** (**var** c **in** chars)  
 **yield return** c;  
 }  
 }  
  
 **protected** PlayFairSettings()  
 {  
 **HsAlphabet** = **new** HashSet<**char**>();  
 **Replaces** = **new** Dictionary<**char**, **char**>();  
   
 *// Переносим символы алфавита в хэштаблицу, чтобы   
 // убрать повторяющиеся значения и быстро определять  
 // принадлежит символ алфавиту или нет* **if** (**AlphabetChars** == **null**) **return**;  
 **foreach** (**var** c **in AlphabetChars**)  
 **HsAlphabet**.Add(c);  
 }  
   
 */// <summary>Возвращает символ после всех преобразований</summary>  
 /// <remarks>Если символ не принадлежит алфавиту, будет поставлен Replacer.</remarks>* **public char** GetChar(**char** Char)  
 {  
 Char = **char**.ToLower(Char);  
 Char = **Replaces**.ContainsKey(Char) ? **Replaces**[Char] : Char;  
 **return HsAlphabet**.Contains(Char) ? Char : **Replacer**;  
 }  
 }  
 **public class** PlayFairRu56 : PlayFairSettings  
 {  
 **public** PlayFairRu56()  
 : **base**()  
 {  
 **Replaces**.Add(**'ё'**, **'е'**);  
 **Replaces**.Add(**'й'**, **'и'**);  
 **Replaces**.Add(**'ъ'**, **'ь'**);  
 }  
   
 **protected override** IEnumerable<**char**> **AlphabetChars** => **"абвгдежзиклмнопрстуфхцчшщьыэюя"**.ToCharArray();  
 **public override char Replacer** => **'х'**;  
 **public override int Columns** => 5;  
 **public override int Rows** => 6;  
 }  
 **public sealed class** PlayFair  
{  
 **struct** TablePosition  
 {  
 **public int Row**;  
 **public int Column**;  
   
 **public** TablePosition(**int** Row, **int** Column)  
 {  
 **this**.**Row** = Row;  
 **this**.**Column** = Column;  
 }  
 }  
  
 **private readonly char**[,] **\_matrix**;  
 **private readonly** Dictionary<**char**, TablePosition> **\_positions**; *// Позиции символов в матрице* **private** PlayFairSettings **Settings** { **get**; set; }  
 **private string Key** { **get**; set; }  
   
 **public** PlayFair(PlayFairSettings Settings, **string** Key)  
 {  
 **this**.**Settings** = Settings;  
 **this**.**Key** = Key;  
   
 *// Формирование матрицы* **\_positions** = **new** Dictionary<**char**, TablePosition>();  
 **var** items = MatrixItems().GetEnumerator();  
 **\_matrix** = **new char**[Settings.**Rows**, Settings.**Columns**];  
 **for** (**var** r = 0; r < Settings.**Rows**; r++)  
 {  
 **for** (**var** c = 0; c < Settings.**Columns**; c++)  
 {  
 **if** (items.MoveNext())  
 {  
 **\_matrix**[r, c] = items.**Current**;  
 **\_positions**.Add(items.**Current**, **new** TablePosition(r, c));  
 }  
 **else throw new** ArgumentException(**"Алфавит слишком маленький"**);  
 }  
 }  
 }  
  
 **private** IEnumerable<**char**> MatrixItems()  
 {  
 **var** used = **new** HashSet<**char**>();  
   
 *// Сначала пишем символы ключа* **foreach** (**char** c **in Key**)  
 {  
 **char** rc = **Settings**.GetChar(c);  
 **if** (used.Contains(rc)) **continue**;  
 used.Add(rc);  
 **yield return** rc;  
 }  
   
 *// Теперь оставшиеся символы алфавита* **foreach** (**char** c **in Settings**.**Alphabet**)  
 {  
 **if** (used.Contains(c)) **continue**;  
 used.Add(c);  
 **yield return** c;  
 }  
 }  
  
   
 *//Разбиаение текста на символы по биграммам.  
 //eplacer выкидывается* **public** IEnumerable<**char**> Bigrams(**string** Text)  
 {  
 **char** prev = **'\0'**; *// Храним перывй символ биграммы* **bool** even = **false**; *// если второй символ биграммы* **foreach** (**char** c **in** Text)  
 {  
 *// Преобразуем символ из текста  
 // он может стать Replacer'ом, если, например, это пробел* **char** rc = **Settings**.GetChar(c);  
  
 **if** (!even) *// Если это первый символ биграммы* {  
 *// запоминаем и ищем второй* prev = rc;  
 even = **true**;  
 }  
 **else** {  
 *// Это второй символ биграммы* **if** (prev == rc) *// и он такойже как и первый* {  
 **if** (prev == **Settings**.**Replacer**) **continue**;  
 *// то вернем биграмму с Replacer в конце* **yield return** prev;  
 **yield return Settings**.**Replacer**;  
  
 *// и будем считать, что уже нашли первый символ в следующей биграмме* prev = rc;  
 }  
 **else** {  
 *// Ну, а если они разные, то вернем оба и будем искать дальше* **yield return** prev;  
 **yield return** rc;  
 even = **false**;  
 }  
  
 }  
 }  
  
 *// Если мы ищем второй символ, а строка закончилась,   
 // при этом первый символ не Replacer* **if** (!even || prev == **Settings**.**Replacer**) **yield break**;  
 **yield return** prev;  
 **yield return Settings**.**Replacer**;  
 }  
  
 **public string** Crypt(**string** text, **bool** modeCrypt = **true**)  
 {  
 **var** shift = modeCrypt ? 1 : -1;  
 **var** sb = **new** StringBuilder();  
 *// Разбиваем на биграммы* **var** chars = Bigrams(text).GetEnumerator();  
 **while** (chars.MoveNext())  
 {  
 *// Получаем координаты символов биграммы в таблице  
 // При расшифровке, если шифротекст неверен   
 // т.е. имеет нечетную длину или неизвестные алфавиту символы  
 // вылетит исключение (поэтому лучше раздить на два метода: шифровки и дешифровки  
 // и перебрасывать исключение). Для шифровки такого не будет* **var** p1 = **\_positions**[chars.**Current**];  
 chars.MoveNext();  
 **var** p2 = **\_positions**[chars.**Current**];  
   
 *// Если они на одной строке - переводим вправо  
 // Если в одной колнке - вниз* **var** error = 0;  
 **if** (p1.**Column** == p2.**Column**)  
 {  
 p1.**Column** = Mod(p1.**Column** + shift, **Settings**.**Columns**);  
 p2.**Column** = Mod(p2.**Column** + shift, **Settings**.**Columns**);  
 error++;  
 }  
 **else if** (p1.**Row** == p2.**Row**)  
 {  
 p1.**Row** = Mod(p1.**Row** + shift, **Settings**.**Rows**);  
 p2.**Row** = Mod(p2.**Row** + shift, **Settings**.**Rows**);  
 error++;  
 }  
   
 **if** (error == 2)  
 **throw new** ArgumentException(**"Неверные биграммы"**);  
   
 sb.Append(**\_matrix**[p1.**Row**, p2.**Column**]);  
 sb.Append(**\_matrix**[p2.**Row**, p1.**Column**]);  
 }  
 **return** sb.ToString();  
 }  
   
 **private static int** Mod(**int** x, **int** m)  
 {  
 *// остаток от деления для отрицательных чисел  
 // для -1 вернет (m-1)* **return** (x % m + m) % m;  
 }  
   
 **public override string** ToString()  
 {  
 **var** sb = **new** StringBuilder();  
 sb.Append(**"Ключ: "**);  
 sb.AppendLine(**Key**);  
 **for** (**int** r = 0; r < **Settings**.**Rows**; r++)  
 {  
 **for** (**int** c = 0; c < **Settings**.**Columns**; c++)  
 {  
 sb.Append(**\_matrix**[r, c]);  
 sb.Append(**' '**);  
 }  
 sb.AppendLine();  
 }  
 **return** sb.ToString();  
 }  
 }  
  
 **internal static class** Program  
 {  
 **private static void** Main(**string**[] args)  
 {  
 **try** {  
 **var** ps = **new** PlayFairRu56();  
 **var** pf = **new** PlayFair(ps, **"Шифр Плейфера"**);  
 Console.WriteLine(pf);  
   
 **var** text = **"Текст который нужно зашифровать шифром Плейферах"**;  
 Console.WriteLine(**"Исходный текст\r\n"** + text);  
   
 **var** i = 0;  
 **foreach** (**var** c **in** pf.Bigrams(text))  
 {  
 Console.Write(c);   
   
 i++;  
 **if** (i % 2 == 0) Console.Write(**' '**);  
 **if** (i % 10 == 0) Console.WriteLine();  
 }  
   
 text = pf.Crypt(text, **true**);  
 Console.WriteLine(**"\r\nЗашифрованный текст\r\n"** + text);  
 }  
 **catch** (Exception ex)  
 {  
 Console.WriteLine(ex.**Message**);  
 }  
 Console.ReadLine();  
 } } }

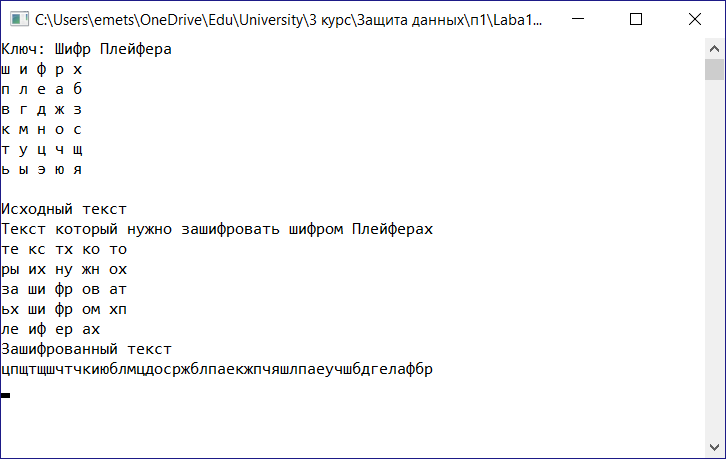


Рис. 5 – шифр Плейфера

**Завдання №6**

Необхідно зашифрувати своє прізвище за допомогою системи оморфонів

У 1401 р Симеона де Крему став використовувати таблиці омофонів для приховування частоти появи голосних букв в тексті за допомогою більш ніж однієї шіфрозамени. Такі шифри пізніше стали називатися шифрами багатозначною заміни або омофонамі2. Вони отримали розвиток в XV столітті. У книзі «Трактат про шифри» Леона Батісти Альберті (італійський вчений, архітектор, теоретик мистецтва, секретар Папи Климентія XII), опублікованій в 1466 р наводиться опис шифру заміни, в якому кожній букві ставиться у відповідність декілька еквівалентів, число яких пропорційно частоті тієї, що зустрічається літери у відкритому тексті. Так, якщо орієнтуватися на табл.4.1, то число шіфрозамен для букви О має становити 110, для букви Е - 85 і т.д. При цьому кожна шіфрозамена повинна складатися з 3 цифр і їх загальна кількість дорівнює 1000. На рис. 6.1 представлений фрагмент таблиці шіфрозамен.

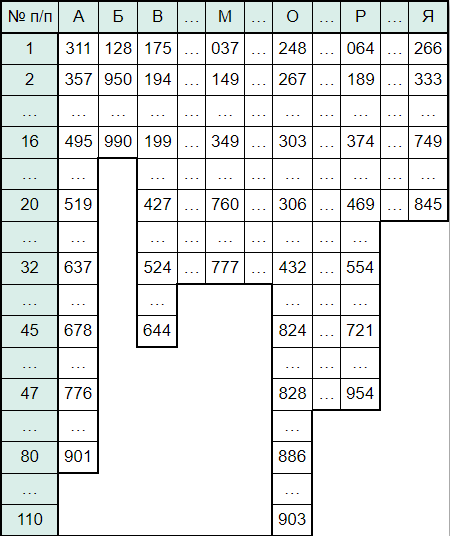


Рис.6.1. Фрагмент таблиці шіфрозамен для системи омофонів

**using** System;  
**namespace** HomophonicReplace  
{  
 **internal static class** Program  
 {  
 **public static void** Main(**string**[] args)  
 {  
 Console.WriteLine(**"Enter word:"**);  
 **var** s1 = Console.ReadLine();  
 **var** s2 = **string**.**Empty**;  
 **if** (s1 != **null**)  
 **foreach** (**var** t **in** s1)  
 {  
 **if** (t != **' '**)  
 **if** (t == 1071)  
 s2 += (**char**) (1103);  
 **else if** (t == 1040)  
 s2 += (**char**) (1071);  
 **else** s2 += (**char**) (t - 1);  
 **else** s2 += t;  
 }  
  
 Console.WriteLine(**"Replaced word {0}"**,s2);  
 Console.ReadKey(**true**);  
 }  
 }  
}

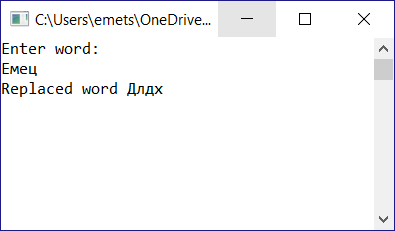


Рис. 6.2 – шифр омофонів

**Завдання №7**

Необхідно зашифрувати своє прізвище за допомогою шифру Віженера

Шифр Віженера - метод поліалфавітних шифрування літерного тексту з використанням ключового слова.

Цей метод є простою формою багатоалфавітної заміни. Шифр Віженера винаходився багаторазово. Метод простий для розуміння і реалізації, він є недоступним для простих методів криптоаналізу.

Хоча шифр легко зрозуміти і реалізувати, протягом трьох століть він пручався всім спробам його зламати; ніж та заробив назву le chiffre indéchiffrable (з французького 'нерозгаданий шифр'). Багато людей намагалися реалізувати схеми шифрування, які по суті були шифрами Віженера.

**using** System;  
  
**namespace** VigenereCiper  
{  
 **internal static class** Program  
 {  
 **private static readonly char**[] **Characters** = { **'А'**, **'Б'**, **'В'**, **'Г'**, **'Д'**, **'Е'**, **'Ё'**, **'Ж'**, **'З'**, **'И'**, **'Й'**, **'К'**, **'Л'**, **'М'**, **'Н'**, **'О'**, **'П'**, **'Р'**, **'С'**, **'Т'**, **'У'**, **'Ф'**, **'Х'**, **'Ц'**, **'Ч'**, **'Ш'**, **'Щ'**, **'Ь'**, **'Ы'**, **'Ъ'**, **'Э'**, **'Ю'**, **'Я'**};  
 **private static readonly int N** = **Characters**.**Length**;  
   
 **private static string** Encode(**string** input, **string** keyword)  
 {  
 input = input.ToUpper();  
 keyword = keyword.ToUpper();  
   
 **var** result = **""**;  
 **var** keywordIndex = 0;  
   
 **foreach** (**var** symbol **in** input)  
 {  
 **var** c = (Array.IndexOf(**Characters**, symbol) +  
 Array.IndexOf(**Characters**, keyword[keywordIndex])) % **N**;  
   
 result += **Characters**[c];  
   
 keywordIndex++;  
   
 **if** ((keywordIndex + 1) == keyword.**Length**)  
 keywordIndex = 0;  
 }  
 **return** result;  
 }  
   
 **public static void** Main(**string**[] args)  
 {  
 Console.WriteLine(**"Enter word to encode:"**);  
 **var** inputEncode = Console.ReadLine();  
 Console.WriteLine(**"Enter keyword:"**);  
 **var** key = Console.ReadLine();  
 Console.WriteLine(**"Зашифрованное слово имеет вид {0}"**,Encode(inputEncode, key));  
   
 }  
 }  
}

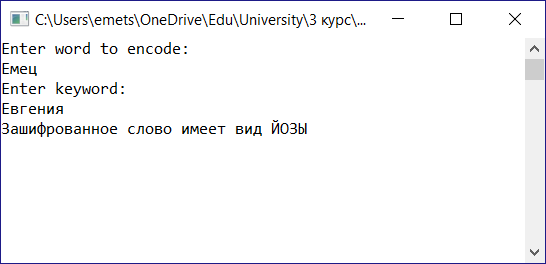


Рис. 7 – шифр Трисемуса