Caesar Cipher

Algorithm

1. Select any integer as key.
2. Treat every letter in the plain text as a number a such that:

```
A = 0, B = 1,... Z = 25
```

3. Shift each letter of plain text up by the number selected as key.

```
E.g., If Key = 2, Character = C, then C + 2 = E
```

4. If the sum exceeds 25 then take mod 26 of the Sum.

```
EncryptedText[i] = (PlainText[i] + Key) mod 26
```

Program

```
class CaesarChiper {
    public string Encrypt(string text, int offset) {
        text = text.ToUpper().Replace(" ", "");
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < text.Length; i++) {
            int absolute = CharUtil.GetAlphaPosition(text[i]);
            int sum = absolute + offset;
            result.Append(CharUtil.GetAlphaFromPosition(sum));
        }
        return result.ToString();
    }

    public string Decrypt(string text, int offset) {
        text = text.ToUpper().Replace(" ", "");
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < text.Length; i++) {
            int absolute = CharUtil.GetAlphaPosition(text[i]);
            int diff = absolute - offset;
            result.Append(CharUtil.GetAlphaFromPosition(diff));
        }
        return result.ToString();
    }
}
```

Output

```
CHIRAG@CHIRAG-DESK MINGW64 ~/Desktop/Cryptography/Cryptography/bin/Debug
$ ./cryptography.exe
Text: CHIRAG SINGH RAJPUT
Key: 5
Encrypted: HMNWFLXNSLMWFOUZY
Decrypted: CHIRAGSINGHRAJPUT
```

Playfair Cipher

Algorithm

Creation And Population Of Matrix

- Enter the keyword in the matrix row wise left to right and the top to bottom.
- Fill remaining spaces in matrix with rest of English alphabets(A-Z) that were not part of our keyword . While doing so combine 'I' and 'J' in the same cell of the table.

Encryption process

- The plain text is broken into groups of 2 alphabets.
- If both the alphabets in pair are same or only 1 is left, add an X after the 1st alphabet.
- If both the alphabets in pair appear in same row of the matrix , replace them with alphabets to their immediate right respectively. If the original player is on right side of row then wrapping around to left side of row.
- If both the alphabets in pair appear in same column of the matrix , replace them with alphabets to their immediate bottom respectively. If the original player is on bottom side of row then wrapping around to top side of row.
- If both the alphabets in pair do not appear in same row or column of the matrix , replace them with alphabets in the same row.

Program

```
class PlayFair {
    string key;
    char[,] matrix = new char[5, 5];

    public string Key { get { return key; } }
    public char[,] Matrix { get { return matrix; } }
```

```
public PlayFair(string key) {
    this.key = key.ToUpper().Replace(" ", "");
    string queryString;

    if (this.Key.Contains('J') && this.Key.Contains('I')) {
        queryString = this.Key.Replace('J', 'I') + "ABCDEFGHKLMNOPQRSTUVWXYZ";
    } else if (this.Key.Contains('J')) {
        queryString = this.Key.Replace("J", "") + "ABCDEFGHIKLMNOPQRSTUVWXYZ";
    } else {
        queryString = this.Key + "ABCDEFGHIKLMNOPQRSTUVWXYZ";
    }

    List<char> characters = queryString
                .Select(character => character)
                .Distinct()
                .ToList();

    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            matrix[i, j] = characters[i * 5 + j];
        }
    }
}

public string Encrypt(string word) {
    List<string> words = SplitText(word);
    StringBuilder result = new StringBuilder();
    foreach (string pair in words) {
        var c1 = MatrixUtil.GetIndex<char>(pair[0], matrix);
        var c2 = MatrixUtil.GetIndex<char>(pair[1], matrix);
        if (c1.Item1 == c2.Item1) {             // SAME ROW
            result.Append(matrix[c1.Item1, (c1.Item2 + 1) % 5]);
            result.Append(matrix[c2.Item1, (c2.Item2 + 1) % 5]);
        } else if (c1.Item2 == c2.Item2) {      // SAME COLUMN
            result.Append(matrix[(c1.Item1 + 1) % 5, c1.Item2]);
            result.Append(matrix[(c2.Item1 + 1) % 5, c2.Item2]);
        } else {                                // DIAGONAL
            result.Append(matrix[c1.Item1, c2.Item2]);
            result.Append(matrix[c2.Item1, c1.Item2]);
        }
        result.Append(" ");
    }
    return result.ToString();
}

public List<string> SplitText(string word) {
    word = word.ToUpper().Replace(" ", "").Replace('J', 'I');
    List<string> splittedText = new List<string>();
    StringBuilder temp = new StringBuilder();
    for (int i = 0; i < word.Length; i++) {
```

```
            if (temp.Length == 2) {
                if ((i - 3) > 0 && word[i - 3] == temp[0]) {
                    temp[1] = temp[0]; temp[0] = 'X';
                    i--;
                } else if (temp[0] == temp[1]) {
                    temp[1] = 'X';
                    i--;
                }
                splittedText.Add(temp.ToString());
                temp.Clear();
            }
            temp.Append(word[i]);
        }

        if (temp.Length >= 1) {
            temp.Append('X');
            splittedText.Add(temp.ToString());
        }
        return splittedText;
    }
}
```

Output

```
CHIRAG@CHIRAG-DESK MINGW64 ~/Desktop/Cryptography/Cryptography/bin/Debug
$ ./cryptography.exe
Key: HARSH
Text: MY NAME IS JUI KAHATE I AM HARSHUHS SISTER
Matrix:

 H  A  R  S  B
 C  D  E  F  G
 I  K  L  M  N
 O  P  Q  T  U
 V  W  X  Y  Z

Splitting: MY NA ME IS IU IK AH AT EI AM HA RS HU HS XS IS TE RX
Encrypted: TS KB LF MH NO KL RA SP CL SK AR SB BO AB YR MH QF ER
```

Hill Cipher

Algorithm

- Treat every alphabet in the plain text as a number. **(A=0,B=1,.....Y=24,Z=25)**

- The plaintext is organized as a matrix of numbers.
- Plaintext matrix is multiplied by a matrix of randomly chosen keys. The key matrix consist of size n*n, where n is the no. of rows and columns in our plaintext matrix.
- Multiply the 2 matrices.
- Compute a mod 26 value of the above matrix.
- Translate the numbers to alphabets.

Program

```
class HillCipher {
    public string Encrypt(string text, int[,] key) {
        if ((key.GetLength(0) != key.GetLength(1)) || text.Length != key.GetLength(0)) {
            throw new ArgumentException("Matrix should of size NxN");
        }

        int[,] textMat = new int[text.Length, 1];

        for (int i = 0; i < text.Length; i++) {
            textMat[i,0] = CharUtil.GetAlphaPosition(text[i]);
        }

        int[,] mult = MatrixUtil.Mult(key, textMat);
        mult = MatrixUtil.Map(mult, x => x % 26);

        StringBuilder result = new StringBuilder();

        for (int i = 0; i < text.Length; i++) {
            result.Append(CharUtil.GetAlphaFromPosition(mult[i, 0]));
        }
        return result.ToString();
    }

    public string Decrypt(string text, int[,] key) {

        int determinant  =  MatrixUtil.Determinant(key);
        determinant = determinant < 0? 26 - (Math.Abs(determinant) % 26) : determinant % 26;

        int inverseDeterminant = 0;
        for (int i = 1; i < 26; i++) {
            if ((i * determinant) % 26 == 1) {
                inverseDeterminant = i;
                break;
            }
        }

        key = MatrixUtil.Adjugate(key);
        key = MatrixUtil.Map(key, x => x < 0 ? 26 - (Math.Abs(x) % 26) : x % 26);
```

```
        key = MatrixUtil.Map(key, x => x * inverseDeterminant);
        key = MatrixUtil.Map(key, x => x % 26);


        int[,] textMat = new int[text.Length, 1];

        for (int i = 0; i < text.Length; i++) {
            textMat[i, 0] = CharUtil.GetAlphaPosition(text[i]);
        }

        StringBuilder result = new StringBuilder();

        int[,] mult = MatrixUtil.Mult(key, textMat);
        mult = MatrixUtil.Map(mult, x => x % 26);

        for (int i = 0; i < text.Length; i++) {
            result.Append(CharUtil.GetAlphaFromPosition(mult[i, 0]));
        }

        return result.ToString();

    }
}
```

Output

```
CHIRAG@CHIRAG-DESK MINGW64 ~/Desktop/Cryptography/Cryptography/bin/Debug
$ ./cryptography.exe
Hill Cipher


Key Matrix:

6       24      1
13      16      10
20      17      15

Text: CAT
Encrypted: FIN
Decrypted: CAT
```

Rail fence Cipher

Algorithmn

- Write the message letters out diagonally over a number of rows.
- Read cipher row by row.

Program

```
class RailFence {
    public string Encrypt(string text) {
        StringBuilder left = new StringBuilder();
        StringBuilder right = new StringBuilder();
        for (int i = 0; i < text.Length; i++) {
            if (i % 2 == 0) {
                left.Append(text[i]);
            } else {
                right.Append(text[i]);
            }
        }
        return left.Append(right.ToString()).ToString();
    }

    public string Decrypt(string text) {
        StringBuilder result = new StringBuilder();
        int offset = (text.Length % 2 == 0) ? 0 : 1;
        int max = text.Length / 2;
        for (int i = 0; i < max; i++) {
            result.Append(text[i]).Append(text[max + i + offset]);
        }
        if (offset > 0) result.Append(text[max + 1]);
        return result.ToString();
    }
}
```

Output

```
CHIRAG@CHIRAG-DESK MINGW64 ~/Desktop/Cryptography/Cryptography/bin/Debug
$ ./cryptography.exe
Rail Fence


Text: COME HOME TOMORROW
Encrypted: CM OETMROOEHM OORW
Decrypted: COME HOME TOMORROW
```

Simple Column Transposition

- Write the plaintext message row by row in a rectangle of predefined size.
- Read the message column by column. It can be any random order.
- The message obtained is the cipher text.

Program

```
class SimpleColumnTransposition {
    char[][] matrix;
    int innerDimension;
    int[] sequence;
    public SimpleColumnTransposition(int dimension, params int[] sequence) {
        for (int i = 1; i <= dimension; i++) {
            if (!sequence.Contains(i))
                throw new Exception("Sequnce must contain all the columns");
        }

        this.sequence = sequence;
        matrix = new char[dimension][];
        innerDimension = dimension;
    }
    public string Encrypt(string text, int rounds = 1) {
        text = text.Replace(" ", "");
        if (rounds == 0) return text;

        int outerDimension = (int)Math.Ceiling((double)text.Length / innerDimension);
        for (int x = 0; x < innerDimension; x++) {
            matrix[x] = new char[outerDimension];
    }

        int i = 0, j = 0, index = 0;
        while (index < text.Length) {
            matrix[i++][j] = text[index++];
            if (i == innerDimension) {
                i = 0; j++;
            }
        }

        StringBuilder result = new StringBuilder();
        for (i = 0; i < sequence.Length; i++) {
            for (j = 0; j < outerDimension; j++) {
                if (matrix[sequence[i] - 1][j] != '\0') {
                    result.Append(matrix[sequence[i] - 1][j]);
                }
            }
        }
        return Encrypt(result.ToString(), rounds - 1);
```

```
    }
}
```

Output

```
CHIRAG@CHIRAG-DESK MINGW64 ~/Desktop/Cryptography/Cryptography/bin/Debug
$ ./cryptography.exe
Simple Column Transposition w/ Multiple Rounds


Text: COME HOME TOMORROW
Number Of Columns: 6
Sequence: 1 5 3 4 6 2
One Round : CMRHMMTOEOWOOOER
Two Rounds: CTOMWREEHORMOMOO
Three Rounds: CEOWROHOMOORMTEM
```

Vernam Cipher

- Treat every plaintext alphabet in the as a number.(A=0,B=1,.....Y=24,Z=25).
- Do the same for each of the alphabet in the input cipher text.
- Add each no corresponding to the plaintext alphabet to the corresponding input cipher text alphabet no.
- If the sum produced is greater than 26, subtract 26 from it.
- Translate each number of the sum back to their corresponding alphabet. This gives the output cipher text.

Program

```
class VernamCipher {
    public string Encrypt(string text, string oneTimePad) {

        text = text.ToUpper().Replace(" ", "");
        oneTimePad = oneTimePad.ToUpper().Replace(" ", "");
        if (text.Length != oneTimePad.Length) return "";

        StringBuilder result = new StringBuilder();

        for (int i = 0; i < text.Length; i++) {
            int sum = CharUtil.GetAlphaPosition(text[i])
                    + CharUtil.GetAlphaPosition(oneTimePad[i]);
```

```
            char c = CharUtil.GetAlphaFromPosition(sum);
            result.Append(c);

        }
        return result.ToString();
    }

    public string Decrypt(string text, string oneTimePad) {
        text = text.ToUpper().Replace(" ", "");
        oneTimePad = oneTimePad.ToUpper().Replace(" ", "");
        if (text.Length != oneTimePad.Length) return "";

        StringBuilder result = new StringBuilder();
        for (int i = 0; i < text.Length; i++) {
            int sum = CharUtil.GetAlphaPosition(text[i])
                    - CharUtil.GetAlphaPosition(oneTimePad[i]);
            char c = CharUtil.GetAlphaFromPosition(sum);
            result.Append(c);

        }
        return result.ToString();
    }
}
```

Output

```
CHIRAG@CHIRAG-DESK MINGW64 ~/Desktop/Cryptography/Cryptography/bin/Debug
$ ./cryptography.exe
Vernam Cipher


Text: HOW ARE YOU
One Time Pad: NC BTZQ ARX
Encrypted: UQXTQUYFR
Decrypted: HOWAREYOU
```

Diffie Hellman Key Exchange

Algorithm

- Mr X and Mr Y agree on two prime numbers n and g.
- Mr X chooses another random number x and calculate $A=g^x \% n$.
- Mr X sends the number A to Mr Y.

- Mr Y chooses another random number y and calculate $B=g^y \% n$.
- Mr Y sends the number B to Mr X.
- A now computes the secret key K1 : $K1=B^x \% n$
- B now computes the secret key K1 : $K1=A^y \% n$

Program

```
ublic class DeffieHellmanKeyExchange {
    public void GenerateKeys(
                int n, int g, int x, out int y,
                out int a, out int b, out int k1, out int k2) {

        y = Enumerable.Range(2, 10)
                .Where(num => MathUtil
                .IsPrime(num))
                .Select(num => num)
                .ToList()
                .Random();

        a = (int)Math.Pow(g, x) % n;
        b = (int)Math.Pow(g, y) % n;
        k1 = (int)Math.Pow(b, x) % n;
        k2 = (int)Math.Pow(a, y) % n;
    }
}
```

Output

```
CHIRAG@CHIRAG-DESK MINGW64 ~/Desktop/Cryptography/Cryptography/bin/Debug
$ ./cryptography.exe
Deffie Hellman Key Exchange


N: 11, G: 7
X: 3, Y: 5
A: 2, B: 10
K1: 10, K2: 10
```

Stream Cipher

Algorithm

- Convert the plaintext(which is in text format i.e, ASCII value) to binary value.
- The Binary Key to be applied must be of atmost 8 bytes.
- XOR operation is applied between the plaintext and the key(1 bit of key for every respective bit of the plain text).
- The result obtained is the cipher text which is in binary format, which when translated to ASCII makes no sense.

Program

```
public class StreamCipher {
    public string Encrypt(string text, byte key) {
        char[] result = new char[text.Length];
        for (int i = 0; i < text.Length; i++) {
            result[i] = (char)(text[i] ^ key);
        }

        return new String(result);
    }

    public string Decrypt(string text, byte key) {
        return Encrypt(text, key);
    }
}
```

Output

```
CHIRAG@CHIRAG-DESK MINGW64 ~/Desktop/Cryptography/Cryptography/bin/Debug
$ ./cryptography.exe
Stream Cipher


Text: PAY100
Key: 12
Encrypted: \MU=<<
Decrypted: PAY100
```

Block Cipher

Algorithm

- Convert the plaintext(which is in text format i.e, ASCII value) to binary value.

- The Binary Key to be applied must be of atmost 8 bytes.
- XOR operation is applied between the plaintext and the key(Unlike Stream cipher here, one block of characters gets encrypted at a time).
- The result obtained is the cipher text which is in binary format, which when translated to ASCII makes no sense.

Program

```
public class BlockCipher {

    public string Decrypt(string text, byte[] keyBytes) {
        return Encrypt(text, keyBytes);
    }

    public string Encrypt(string text, byte[] keyBytes) {

        if (keyBytes.Length > 8)
            throw new ArgumentException("Key Can't Be Greater Than 64 Bits");

        int keyLength = keyBytes.Length;
        List<string> textBlocks = SplitText(text, keyLength);

        char[] encrypted = new char[text.Length];

        for (int i = 0; i < textBlocks.Count; i++) {
            for (int j = 0; j < keyLength; j++) {
                int index = i * keyLength + j;
                if (index < encrypted.Length) {
                    encrypted[index] = (char)(textBlocks[i][j] ^ keyBytes[j]);
                }
            }
        }

        return new String(encrypted);

    }

    public List<string> SplitText(string text, int blockSize) {
        StringBuilder result = new StringBuilder();
        List<string> splittedText = new List<string>();
        for (int i = 0; i < text.Length; i++) {
            if (result.Length == blockSize) {
                splittedText.Add(result.ToString());
                result.Clear();
            }
            result.Append(text[i]);
        }
        splittedText.Add(result.ToString());
```

```
        return splittedText;
    }
}
```

Output

```
CHIRAG@CHIRAG-DESK MINGW64 ~/Desktop/Cryptography/Cryptography/bin/Debug
$ ./cryptography.exe
Block Cipher


Key: 1 4 9
Text: FOUR_AS_FOUR
Encrypted: GK\S[HR[ONQ[
Decrypted: FOUR_AS_FOUR
```

Helper Functions

Character Utility Class

```
public static class CharUtil {
    public static int GetAlphaPosition(char c) {
        if (c >= 97 && c <= 122) return c - 97;
        else if (c >= 65 && c <= 90) return c - 65;
        else return -1;
    }

    public static char GetAlphaFromPosition(int pos) {
        if (pos > 25) return (char)((pos % 26) + 65);
        else if (pos < 0) return (char)(((pos + 26) % 26) + 65);
        else return (char)(pos + 65);
    }

    public static char NextChar(char c) {
        if (c == 'Z') return 'A';
        else if (c == 'z') return 'a';
        else return (char)(c + 1);
    }
}
```

List Extension Methods

```
static class ListExtensions {
    public static T Random<T>(this List<T> obj) {
        Random rand = new Random();
        return obj[rand.Next(obj.Count)];
    }
}
```

Matrix Utility Class

```
public static class MatrixUtil {
    public static Tuple<int, int> GetIndex<T>(T c, T[,] matrix) {
        for (int i = 0; i < matrix.GetLength(0); i++) {
            for (int j = 0; j < matrix.GetLength(1); j++) {
                if (matrix[i, j].Equals(c)) {
                    return Tuple.Create<int, int>(i, j);
                }
            }
        }
        return Tuple.Create<int, int>(-1, -1);
    }

    public static T[,] Mult<T>(T[,] m1, T[,] m2) {

        int r1 = m1.GetLength(0), r2 = m2.GetLength(0);
        int c1 = m1.GetLength(1), c2 = m2.GetLength(1);

        if (c1 != r2) {
            throw new ArgumentException("Columns Of M1 != Rows Of M2");
        }

        T[,] result = new T[r1, c2];

        for (int i = 0; i < r1; i++) {
            for (int j = 0; j < c2; j++) {
                T sum = default(T);
                for (int k = 0; k < c1; k++) {
                    sum += (dynamic)m1[i, k] * m2[k, j];
                }
                result[i, j] = sum;
            }
        }
        return result;
    }

    public static T[,] Map<T>(T[,] matrix, Func<T, T> func) {
        int rows = matrix.GetLength(0), cols = matrix.GetLength(1);
        T[,] output = new T[rows, cols];
```

```
    for (int i = 0; i < matrix.GetLength(0); i++) {
        for (int j = 0; j < matrix.GetLength(1); j++) {
            output[i, j] = func(matrix[i, j]);
        }
    }
    return output;
}

public static T[,] Transpose<T>(T[,] matrix) {
    int rows = matrix.GetLength(0), cols = matrix.GetLength(1);
    T[,] transpose = new T[rows, cols];
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            transpose[i, j] = matrix[j, i];
        }
    }
    return transpose;
}


public static T[,] Inverse<T>(T[,] matrix) {
    T[,] adj = Adjugate<T>(matrix);
    T determinant = Determinant<T>(matrix);
    return Map<T>(adj, x => (dynamic)x / determinant);
}

public static T[,] Adjugate<T>(T[,] matrix) {
    int rows = matrix.GetLength(0), cols = matrix.GetLength(1);
    T[,] adj = new T[rows, cols];

    List<T[,]> minors = Minors<T>(matrix);
    for (int i = 0; i < rows; i++) {
        int sign = i % 2;
        for (int j = 0; j < cols; j++) {
            adj[i, j] = (dynamic)Determinant<T>(minors[i * rows + j])
                * (int)Math.Pow(-1, sign++);
        }
    }
    return Transpose<T>(adj);
}

public static T Determinant<T>(T[,] matrix) {
    int rows = matrix.GetLength(0), cols = matrix.GetLength(1);
    if (rows != cols)
            throw new Exception("Determinant only exists for square matrices");
    if (rows == 1) return matrix[0, 0];
    if (rows == 2)
            return (dynamic)matrix[0, 0] * matrix[1, 1]
                - (dynamic)matrix[0, 1] * matrix[1, 0];
    else {
        T determinant = default(T);
```

```
            for (int i = 0; i < cols; i++) {
                determinant += (dynamic)(int)Math.Pow(-1, i)
                            * Determinant<T>(SubMatrix<T>(matrix, 0, i)) * matrix[0, i];
            }
            return determinant;
        }
    }

    public static List<T[,]> Minors<T>(T[,] matrix) {
        int rows = matrix.GetLength(0), cols = matrix.GetLength(1);
        List<T[,]> minors = new List<T[,]>();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                T[,] minor = SubMatrix<T>(matrix, i, j);
                minors.Add(minor);
            }
        }
        return minors;
    }

    public static T[,] SubMatrix<T>(T[,] matrix, int y, int x) {
        int rows = matrix.GetLength(0), cols = matrix.GetLength(1);
        T[,] subMatrix = new T[rows - 1, cols - 1];
        int k = 0, l = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (i != y && j != x) {
                    subMatrix[k, l] = matrix[i, j];
                    if ((l = (l + 1) % (cols - 1)) == 0) k++;
                }
            }
        }
        return subMatrix;
    }
}
```