Name - Khushi Nitinkumar Patel
PRN - 2020BTECS00037
Batch - B3

Assignment no 1: Implementation of Caesar Cipher

Introduction

The Caesar Cipher technique is one of the earliest and simplest methods of encryption technique. It’s simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with a fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on.

Encryption

Here is an example of how to use the Caesar cipher to encrypt the message **“HELLO”** with a shift of 3:

Write down the plaintext message: HELLO
Choose a shift value. In this case, we will use a shift of 3.
Replace each letter in the plaintext message with the letter that is three positions to the right in the alphabet.

1. H becomes K (shift 3 from H)
2. E becomes H (shift 3 from E)
3. L becomes O (shift 3 from L)
4. L becomes O (shift 3 from L)
5. O becomes R (shift 3 from O)

The encrypted message is now **“KHOOR”.**

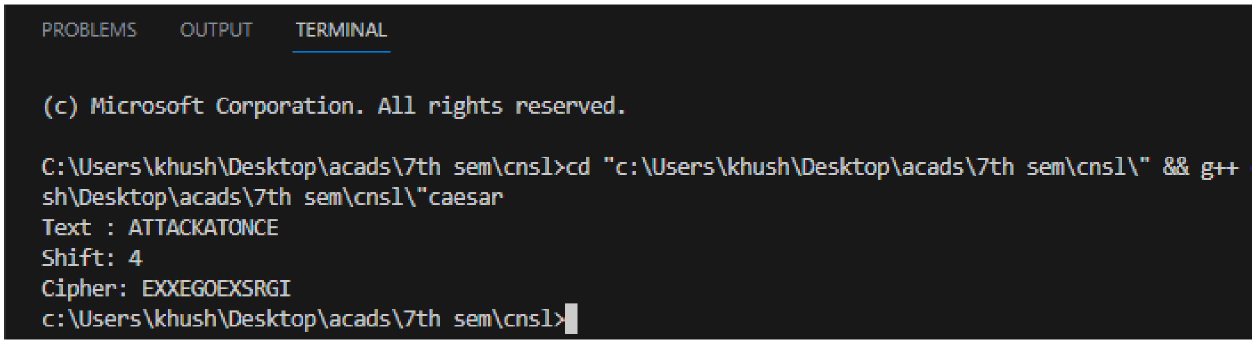
Decryption

To decrypt the message, you simply need to shift each letter back by the same number of positions. In this case, you would shift each letter in **“KHOOR”** back by 3 positions to get the original message, **“HELLO”.**

Encryption Code:

```cpp
#include <iostream>
using namespace std;

// This function receives text and shift and
// returns the encrypted text
string encrypt(string text, int s)
{
    string result = "";
    // traverse text
    for (int i = 0; i < text.length(); i++) {
        // apply transformation to each character
        // Encrypt Uppercase letters
        if (isupper(text[i]))
            result += char(int(text[i] + s - 65) % 26 + 65);

        // Encrypt Lowercase letters
        else
            result += char(int(text[i] + s - 97) % 26 + 97);
    }
    // Return the resulting string
    return result;
}
// Driver program to test the above function
int main()
{
    string text = "ATTACKATONCE";
    int s = 4;
    cout << "Text : " << text;
    cout << "\nShift: " << s;
    cout << "\nCipher: " << encrypt(text, s);
    return 0;
}
```

Output:

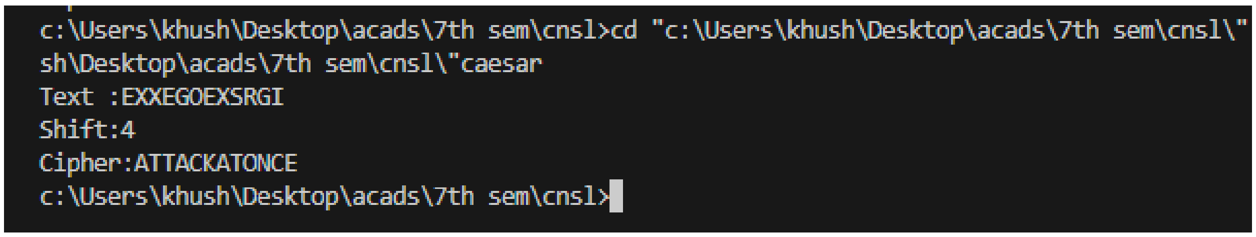


```
(c) Microsoft Corporation. All rights reserved.

C:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Desktop\acads\7th sem\cnsl\" && g++
sh\Desktop\acads\7th sem\cnsl\"caesar
Text : ATTACKATONCE
Shift: 4
Cipher: EXXEGOEXSRGI
c:\Users\khush\Desktop\acads\7th sem\cnsl>
```

Decryption Code:

```cpp
#include <iostream>
using namespace std;

//This function receives text and shift and returns the encrypted text
string encrypt(string text,int s)
{
    string result="";
    //traverse text
    for(int i=0;i<text.length();i++)
    {
        //apply transformation to each character
        //Encrypt Uppercase letters
        if(isupper(text[i]))
            result+=char(int(text[i]+s-65)%26 +65);
    //Encrypt Lowercase letters
    else
        result+=char(int(text[i]+s-97)%26 +97);
    }
    //Return the resulting string
    return result;
}

//Driver program to test the above function
int main()
{
    string text="EXXEGOEXSRGI";
    int s = 4;


    cout<<"Text :"<<text;
    cout<<"\nShift:" << s;
    s = s%26; // ensuring that s lies between 0-25
    cout<<"\nCipher:"<<encrypt(text, 26-s);
    return 0;
}
```

Output:



```
c:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Desktop\acads\7th sem\cnsl\"
sh\Desktop\acads\7th sem\cnsl\"caesar
Text :EXXEGOEXSRGI
Shift:4
Cipher:ATTACKATONCE
c:\Users\khush\Desktop\acads\7th sem\cnsl>
```

Name - Khushi Nitinkumar Patel
PRN - 2020BTECS00037
Batch - B3

Assignment no 2: Implementation of Transposition Cipher

Introduction

a) **Columnar Transposition**
The Columnar Transposition Cipher is a form of transposition cipher just like Rail Fence Cipher. Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one.

Encryption

In a transposition cipher, the order of the alphabets is rearranged to obtain the cipher-text.

1. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
2. Width of the rows and the permutation of the columns are usually defined by a keyword.
3. For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be “3 1 2 4”.
4. Any spare spaces are filled with nulls or left blank or placed by a character (Example: _).
5. Finally, the message is read off in columns, in the order specified by the keyword.

Decryption

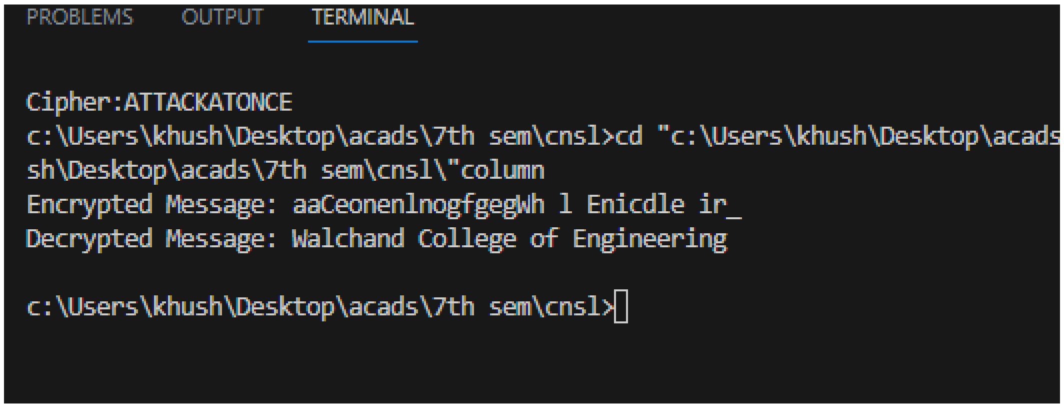
1. To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.
2. Then, write the message out in columns again, then re-order the columns by reforming the key word.

Encryption and Decryption Code:

```cpp
// CPP program for illustrating
// Columnar Transposition Cipher
#include<bits/stdc++.h>
using namespace std;

// Key for Columnar Transposition
string const key = "HACK";
map<int,int> keyMap;

void setPermutationOrder()
{
    // Add the permutation order into map
    for(int i=0; i < key.length(); i++)
    {
        keyMap[key[i]] = i;
    }
}

// Encryption
string encryptMessage(string msg)
{
    int row,col,j;
    string cipher = "";

    /* calculate column of the matrix*/
    col = key.length();

    /* calculate Maximum row of the matrix*/
    row = msg.length()/col;

    if (msg.length() % col)
        row += 1;

    char matrix[row][col];

    for (int i=0,k=0; i < row; i++)
    {
        for (int j=0; j<col; )
        {
            if(msg[k] == '\0')
            {
                /* Adding the padding character '_' */
                matrix[i][j] = '_';
                j++;
            }

            if( isalpha(msg[k]) || msg[k]==' ')
            {
                /* Adding only space and alphabet into matrix*/
                matrix[i][j] = msg[k];
                j++;
            }
            k++;
        }
    }

    for (map<int,int>::iterator ii = keyMap.begin(); ii!=keyMap.end();
++ii)
    {
        j=ii->second;

        // getting cipher text from matrix column wise using permuted key
        for (int i=0; i<row; i++)
        {
            if( isalpha(matrix[i][j]) || matrix[i][j]==' ' ||
matrix[i][j]=='_')
                cipher += matrix[i][j];
        }
    }

    return cipher;
}

// Decryption
string decryptMessage(string cipher)
{
    /* calculate row and column for cipher Matrix */
    int col = key.length();
    int row = cipher.length()/col;
    char cipherMat[row][col];

    /* add character into matrix column wise */
    for (int j=0,k=0; j<col; j++)
        for (int i=0; i<row; i++)
            cipherMat[i][j] = cipher[k++];

    /* update the order of key for decryption */
    int index = 0;
    for( map<int,int>::iterator ii=keyMap.begin(); ii!=keyMap.end(); ++ii)
        ii->second = index++;

    /* Arrange the matrix column wise according
    to permutation order by adding into new matrix */
    char decCipher[row][col];
    map<int,int>::iterator ii=keyMap.begin();
    int k = 0;
    for (int l=0,j; key[l]!='\0'; k++)
    {
        j = keyMap[key[l++]];
        for (int i=0; i<row; i++)
        {
            decCipher[i][k]=cipherMat[i][j];
        }
    }

    /* getting Message using matrix */
    string msg = "";
    for (int i=0; i<row; i++)
    {
        for(int j=0; j<col; j++)
        {
            if(decCipher[i][j] != '_')
                msg += decCipher[i][j];
        }
    }
    return msg;
}
```

```cpp
// Driver Program
int main(void)
{
    /* message */
    string msg = "Walchand College of Engineering";

    setPermutationOrder();

    // Calling encryption function
    string cipher = encryptMessage(msg);
    cout << "Encrypted Message: " << cipher << endl;

    // Calling Decryption function
    cout << "Decrypted Message: " << decryptMessage(cipher) << endl;

    return 0;
}
```

Output:



```
Cipher:ATTACKATONCE
c:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Desktop\acads
sh\Desktop\acads\7th sem\cnsl\"column
Encrypted Message: aaCeonenlnogfgegWh l Enicdle ir_
Decrypted Message: Walchand College of Engineering

c:\Users\khush\Desktop\acads\7th sem\cnsl>
```

b) Rail Fence cipher

The rail fence cipher (also called a zigzag cipher) is a form of transposition cipher. It derives its name from the way in which it is encoded**.**

Encryption

In a transposition cipher, the order of the alphabets is rearranged to obtain the cipher-text.

1. In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.
2. When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.
3. After each alphabet has been written, the individual rows are combined to obtain the cipher-text.

Decryption

The number of columns in rail fence cipher remains equal to the length of plain-text message. And the key corresponds to the number of rails.

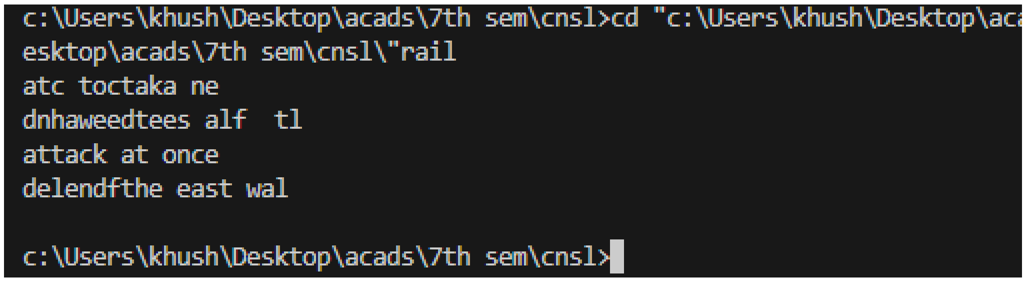
1. Hence, rail matrix can be constructed accordingly. Once we’ve got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively).
2. Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

Encryption and Decryption Code:

```
// C++ program to illustrate Rail Fence Cipher
// Encryption and Decryption
#include <bits/stdc++.h>
using namespace std;

// function to encrypt a message
string encryptRailFence(string text, int key)
{
    // create the matrix to cipher plain text
    // key = rows , length(text) = columns
    char rail[key][(text.length())];

    // filling the rail matrix to distinguish filled
    // spaces from blank ones
    for (int i=0; i < key; i++)
        for (int j = 0; j < text.length(); j++)
            rail[i][j] = '\n';

    // to find the direction
    bool dir_down = false;
    int row = 0, col = 0;

    for (int i=0; i < text.length(); i++)
    {
        // check the direction of flow
        // reverse the direction if we've just
        // filled the top or bottom rail
        if (row == 0 || row == key-1)
            dir_down = !dir_down;

        // fill the corresponding alphabet
        rail[row][col++] = text[i];

        // find the next row using direction flag
        dir_down?row++ : row--;
    }

    //now we can construct the cipher using the rail matrix
    string result;
    for (int i=0; i < key; i++)
        for (int j=0; j < text.length(); j++)
            if (rail[i][j]!='\n')
                result.push_back(rail[i][j]);

    return result;
}

// This function receives cipher-text and key
// and returns the original text after decryption
string decryptRailFence(string cipher, int key)
{
    // create the matrix to cipher plain text
    // key = rows , length(text) = columns
    char rail[key][cipher.length()];

    // filling the rail matrix to distinguish filled
    // spaces from blank ones
    for (int i=0; i < key; i++)
        for (int j=0; j < cipher.length(); j++)
            rail[i][j] = '\n';

    // to find the direction
    bool dir_down;

    int row = 0, col = 0;

    // mark the places with '*'
    for (int i=0; i < cipher.length(); i++)
    {
        // check the direction of flow
        if (row == 0)
            dir_down = true;
        if (row == key-1)
            dir_down = false;

        // place the marker
        rail[row][col++] = '*';

        // find the next row using direction flag
```

```
            dir_down?row++ : row--;
        }

        // now we can construct the fill the rail matrix
        int index = 0;
        for (int i=0; i<key; i++)
            for (int j=0; j<cipher.length(); j++)
                if (rail[i][j] == '*' && index<cipher.length())
                    rail[i][j] = cipher[index++];


        // now read the matrix in zig-zag manner to construct
        // the resultant text
        string result;

        row = 0, col = 0;
        for (int i=0; i< cipher.length(); i++)
        {
            // check the direction of flow
            if (row == 0)
                dir_down = true;
            if (row == key-1)
                dir_down = false;

            // place the marker
            if (rail[row][col] != '*')
                result.push_back(rail[row][col++]);

            // find the next row using direction flag
            dir_down?row++: row--;
        }
        return result;
}

//driver program to check the above functions
int main()
{
    cout << encryptRailFence("attack at once", 2) << endl;
    cout << encryptRailFence("defend the east wall", 3) << endl;
        //Now decryption of the same cipher-text

        cout << decryptRailFence("atc toctaka ne",2) << endl;
        cout << decryptRailFence("dnhaweedtees alf tl",3) << endl;

        return 0;
}
```

Output:



```
c:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Desktop\aca
esktop\acads\7th sem\cnsl\"rail
atc toctaka ne
dnhaweedtees alf  tl
attack at once
delendfthe east wal

c:\Users\khush\Desktop\acads\7th sem\cnsl>
```

Name - Khushi Nitinkumar Patel
PRN - 2020BTECS00037
Batch - B3

Assignment no 3: Implementation of Transposition Cipher

Introduction

In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.

The Playfair Cipher Encryption Algorithm:

The Algorithm consists of 2 steps:

1. **Generate the key Square(5×5):**
   The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I.

   The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

2. **Algorithm to encrypt the plain text:** The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

The Playfair Cipher Decryption Algorithm:

The Algorithm consists of 2 steps:

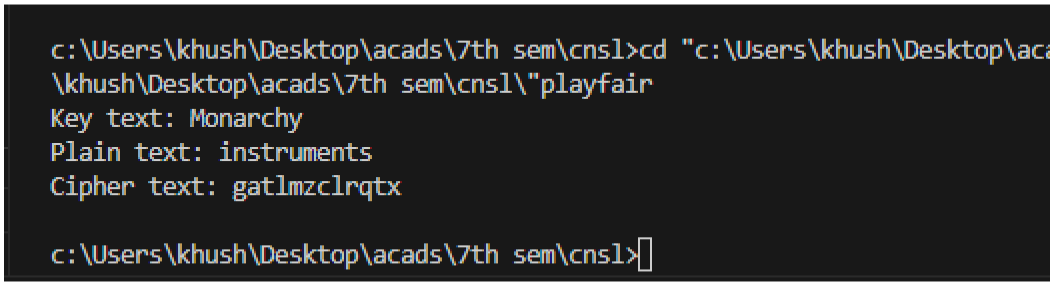
1. **Generate the key Square(5×5) at the receiver's end:**
   The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I.
   The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

2. **Algorithm to decrypt the ciphertext:** The ciphertext is split into pairs of two letters (digraphs).

Encryption Code:

```cpp
// C++ program to implement Playfair Cipher

#include <bits/stdc++.h>
using namespace std;
#define SIZE 30

// Function to convert the string to lowercase
void toLowerCase(char plain[], int ps)
{
    int i;
    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}

// Function to remove all spaces in a string
int removeSpaces(char* plain, int ps)
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

// Function to generate the 5x5 key square
void generateKeyTable(char key[], int ks, char keyT[5][5])
{
    int i, j, k, flag = 0;

    // a 26 character hashmap
    // to store count of the alphabet
    int dicty[26] = { 0 };
    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
            dicty[key[i] - 97] = 2;
    }
    dicty['j' - 97] = 1;

    i = 0;
    j = 0;

    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }

    for (k = 0; k < 26; k++) {
        if (dicty[k] == 0) {
            keyT[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
}

// Function to search for the characters of a digraph
// in the key square and return their position
void search(char keyT[5][5], char a, char b, int arr[])
{
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';
    for (i = 0; i < 5; i++) {

        for (j = 0; j < 5; j++) {

            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}

// Function to find the modulus with 5
int mod5(int a) { return (a % 5); }

// Function to make the plain text length to be even
int prepare(char str[], int ptrs)
{
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}

// Function for performing the encryption
void encrypt(char str[], char keyT[5][5], int ps)
{
    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] + 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] + 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}

// Function to encrypt using Playfair Cipher
void encryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];

    // Key
    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);

    // Plaintext
    ps = strlen(str);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    encrypt(str, keyT, ps);
}

// Driver code
int main()
{
    char str[SIZE], key[SIZE];
    // Key to be encrypted
    strcpy(key, "Monarchy");
    cout << "Key text: " << key << "\n";

    // Plaintext to be encrypted
    strcpy(str, "instruments");
    cout << "Plain text: " << str << "\n";

    // encrypt using Playfair Cipher
    encryptByPlayfairCipher(str, key);

    cout << "Cipher text: " << str << "\n";

    return 0;
}
```

Output:



```
c:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Desktop\aca
\khush\Desktop\acads\7th sem\cnsl\"playfair
Key text: Monarchy
Plain text: instruments
Cipher text: gatlmzclrqtx

c:\Users\khush\Desktop\acads\7th sem\cnsl>
```

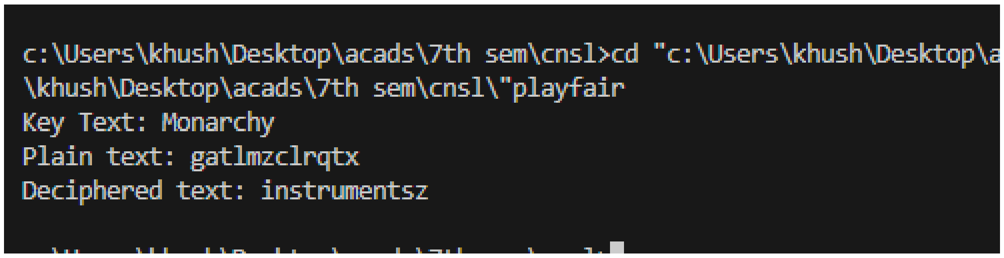
Decryption Code:

```
#include <bits/stdc++.h>
using namespace std;
#define SIZE 30

// Convert all the characters
// of a string to lowercase
void toLowerCase(char plain[], int ps)
{
    int i;
    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}

// Remove all spaces in a string
// can be extended to remove punctuation
int removeSpaces(char* plain, int ps)
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

// generates the 5x5 key square
void generateKeyTable(char key[], int ks, char keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;

    // a 26 character hashmap
    // to store count of the alphabet
    dicty = (int*)calloc(26, sizeof(int));

    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
            dicty[key[i] - 97] = 2;
    }
    dicty['j' - 97] = 1;

    i = 0;
    j = 0;
    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
    for (k = 0; k < 26; k++) {
        if (dicty[k] == 0) {
            keyT[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
}

// Search for the characters of a digraph
// in the key square and return their position
void search(char keyT[5][5], char a, char b, int arr[])
{
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';

    for (i = 0; i < 5; i++) {
```

```c
        for (j = 0; j < 5; j++) {
            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}

// Function to find the modulus with 5
int mod5(int a)
{
    if (a < 0)
        a += 5;
    return (a % 5);
}

// Function to decrypt
void decrypt(char str[], char keyT[5][5], int ps)
{
    int i, a[4];
    for (i = 0; i < ps; i += 2) {
        search(keyT, str[i], str[i + 1], a);
        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] - 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] - 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
```

```
}

// Function to call decrypt
void decryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];

    // Key
    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);

    // ciphertext
    ps = strlen(str);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);

    generateKeyTable(key, ks, keyT);

    decrypt(str, keyT, ps);
}

// Driver code
int main()
{
    char str[SIZE], key[SIZE];

    // Key to be encrypted
    strcpy(key, "Monarchy");
    cout << "Key Text: " << key << endl;

    // Ciphertext to be decrypted
    strcpy(str, "gatlmzclrqtx");
    cout << "Plain text: " << str << endl;

    // encrypt using Playfair Cipher
    decryptByPlayfairCipher(str, key);

    cout << "Deciphered text: " << str << endl;
    return 0;
}
```

Output:



```
c:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Desktop\a
\khush\Desktop\acads\7th sem\cnsl\"playfair
Key Text: Monarchy
Plain text: gatlmzclrqtx
Deciphered text: instrumentsz
```

Name - Khushi Nitinkumar Patel
PRN - 2020BTECS00037
Batch - B3

Assignment no 4: Implementation of Vigenere Cipher

Introduction

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the Vigenère square or Vigenère table.

Encryption:

Suppose we want to encrypt the plaintext message "HELLO" using the keyword "KEY."

1. Key Expansion:

- Repeat the keyword to match the length of the plaintext message:

Plaintext:	H E L L O
Keyword:	K E Y K E

2. Letter to Number Conversion:

- Convert the letters of the plaintext and the keyword to their corresponding numerical values (A=0, B=1, ..., Z=25):

Plaintext:	7 4 11 11 14
Keyword:	10 4 24 10 4

3. Encryption:

- Add the corresponding values of the plaintext and the keyword, taking care to wrap around the alphabet if the sum is greater than 25:

Ciphertext: 17 8 9 21 18

4. Number to Letter Conversion:

- Convert the numerical values back to letters:

Ciphertext: R I J V S

So, "HELLO" encrypted with the keyword "KEY" becomes "RIJVS."

Decryption

Now, let's decrypt the ciphertext "RIJVS" using the same keyword "KEY."

1. Key Expansion:

- Repeat the keyword to match the length of the ciphertext:

```
Ciphertext:   R  I  J  V  S
Keyword:      K  E  Y  K  E
```

2. Letter to Number Conversion:

- Convert the letters of the ciphertext and the keyword to their corresponding numerical values (A=0, B=1, ..., Z=25):

```
Ciphertext:   17 8  9  21 18
Keyword:      10 4  24 10 4
```

3. Decryption:

- Subtract the corresponding values of the keyword from the ciphertext, taking care to wrap around the alphabet if the difference is negative:

```
Plaintext:    7  4  11 11 14
```

4. Number to Letter Conversion:

- Convert the numerical values back to letters:

```
Plaintext:    H  E  L  L  O
```

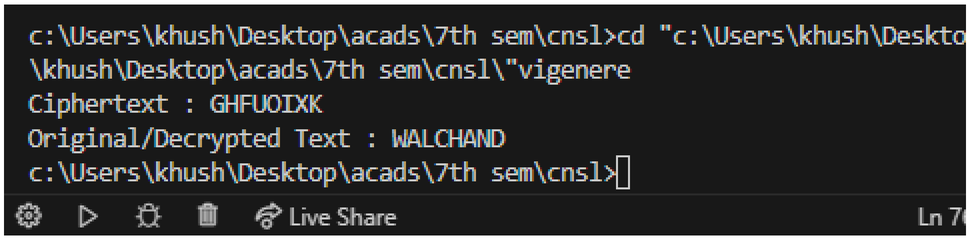
So, "RIJVS" decrypted with the keyword "KEY" becomes "HELLO."

Encryption and Decryption Code:

```
// C++ code to implement Vigenere Cipher
#include<bits/stdc++.h>
using namespace std;

// This function generates the key in
// a cyclic manner until it's length isn't
// equal to the length of original text
string generateKey(string str, string key)
{
    int x = str.size();

    for (int i = 0; ; i++)
    {
        if (x == i)
            i = 0;
        if (key.size() == str.size())
            break;
        key.push_back(key[i]);
    }
    return key;
}

// This function returns the encrypted text
// generated with the help of the key
string cipherText(string str, string key)
{
    string cipher_text;

    for (int i = 0; i < str.size(); i++)
    {
        // converting in range 0-25
        char x = (str[i] + key[i]) %26;

        // convert into alphabets(ASCII)
        x += 'A';

        cipher_text.push_back(x);
    }
    return cipher_text;
}

// This function decrypts the encrypted text
// and returns the original text
string originalText(string cipher_text, string key)
```

```
{
    string orig_text;

    for (int i = 0 ; i < cipher_text.size(); i++)
    {
        // converting in range 0-25
        char x = (cipher_text[i] - key[i] + 26) %26;

        // convert into alphabets(ASCII)
        x += 'A';
        orig_text.push_back(x);
    }
    return orig_text;
}

// Driver program to test the above function
int main()
{
    string str = "WALCHAND";
    string keyword = "KHUSHI";

    string key = generateKey(str, keyword);
    string cipher_text = cipherText(str, key);

    cout << "Ciphertext : "
        << cipher_text << "\n";

    cout << "Original/Decrypted Text : "
        << originalText(cipher_text, key);
    return 0;
}
```

Output:



```
c:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Deskto
\khush\Desktop\acads\7th sem\cnsl\"vigenere
Ciphertext : GHFUOIXK
Original/Decrypted Text : WALCHAND
c:\Users\khush\Desktop\acads\7th sem\cnsl>
```

Name - Khushi Nitinkumar Patel
PRN - 2020BTECS00037
Batch - B3

Assignment no 5: Implementation of DES

Introduction

DES is a block cipher and encrypts data in blocks of size of 64 bits each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.

DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a round. Each round performs the steps of substitution and transposition.

- In the first step, the 64-bit plain text block is handed over to an initial Permutation (IP) function.
- The initial permutation is performed on plain text.
- Next, the initial permutation (IP) produces two halves of the permuted block; saying Left Plain Text (LPT) and Right Plain Text (RPT).
- Now each LPT and RPT go through 16 rounds of the encryption process.
- In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block
- The result of this process produces 64-bit ciphertext.

Implementation

Initial Permutation:

The Initial Permutation (IP) is a one-time operation that occurs before the first round of DES encryption.

IP involves rearranging the bits of the original plaintext block according to a predefined rule.

Each bit in the rearranged block is determined by swapping it with a specific bit from the original plaintext block, as specified by the IP permutation table.

IP is essentially a bit-position juggling operation that establishes the initial data arrangement for subsequent DES rounds.

Step-1: Key transformation:
We have noted initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each a 56-bit key is available. From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called key transformation. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.

Step-2: Expansion Permutation:
Recall that after the initial permutation, we had two 32-bit plain text areas called Left Plain Text(LPT) and Right Plain Text(RPT). During the expansion permutation, the RPT is expanded from 32 bits to 48 bits. Bits are permuted as well hence called expansion permutation. This happens as the 32-bit RPT is divided into 8 blocks, with each block consisting of 4 bits. Then, each 4-bit block of the previous step is then expanded to a corresponding 6-bit block, i.e., per 4-bit block, 2 more bits are added.

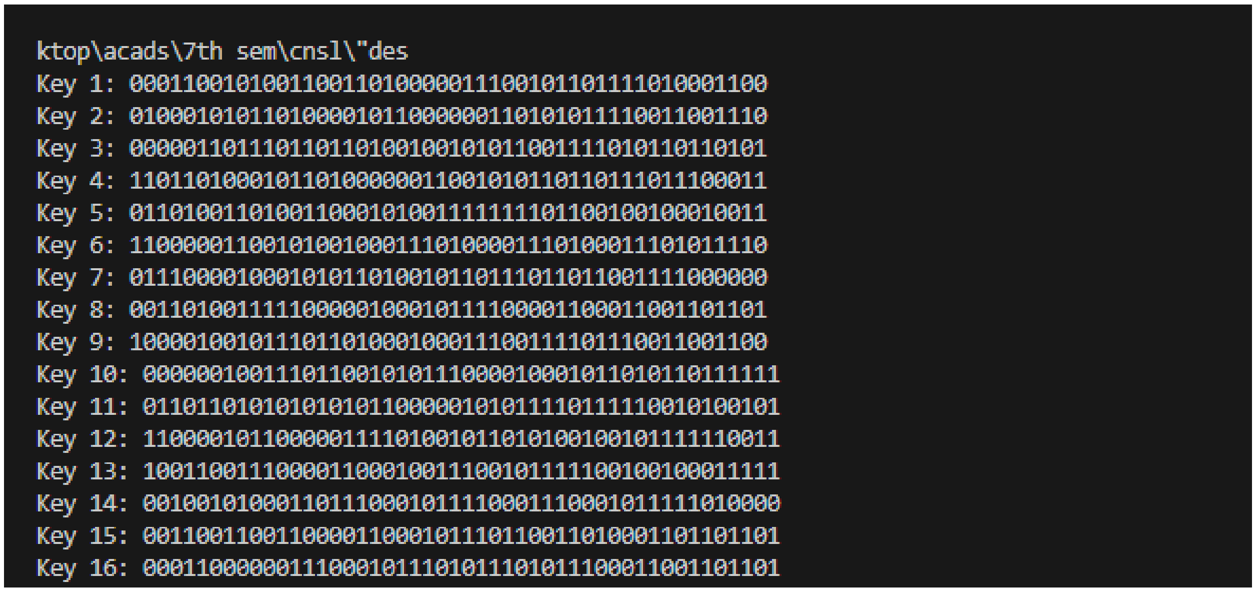
This process results in expansion as well as a permutation of the input bit while creating output. The key transformation process compresses the 56-bit key to 48 bits. Then the expansion permutation process expands the 32-bit RPT to 48-bits. Now the 48-bit key is XOR with 48-bit RPT and the resulting output is given to the next step, which is the S-Box substitution.

1. Generating Keys

```
// Including dependancies
#include <iostream>
#include <string>
using namespace std;
// Array to hold the 16 keys
string round_keys[16];
// Function to do a circular left shift by 1
string shift_left_once(string key_chunk){
    string shifted="";
        for(int i = 1; i < 28; i++){
            shifted += key_chunk[i];
        }
        shifted += key_chunk[0];
    return shifted;
}
// Function to do a circular left shift by 2
string shift_left_twice(string key_chunk){
    string shifted="";
    for(int i = 0; i < 2; i++){
        for(int j = 1; j < 28; j++){
            shifted += key_chunk[j];
        }
        shifted += key_chunk[0];
        key_chunk= shifted;
        shifted ="";
    }
    return key_chunk;
}
void generate_keys(string key){
    // The PC1 table
    int pc1[56] = {
    57,49,41,33,25,17,9,
    1,58,50,42,34,26,18,
    10,2,59,51,43,35,27,
    19,11,3,60,52,44,36,
    63,55,47,39,31,23,15,
    7,62,54,46,38,30,22,
    14,6,61,53,45,37,29,
21,13,5,28,20,12,4
};
// The PC2 table
int pc2[48] = {
14,17,11,24,1,5,
3,28,15,6,21,10,
23,19,12,4,26,8,
16,7,27,20,13,2,
41,52,31,37,47,55,
30,40,51,45,33,48,
44,49,39,56,34,53,
46,42,50,36,29,32
};
// 1. Compressing the key using the PC1 table
string perm_key ="";
for(int i = 0; i < 56; i++){
    perm_key+= key[pc1[i]-1];
}
// 2. Dividing the result into two equal halves
string left= perm_key.substr(0, 28);
string right= perm_key.substr(28, 28);
// Generating 16 keys
for(int i=0; i<16; i++){
    // 3.1. For rounds 1, 2, 9, 16 the key_chunks
    // are shifted by one.
    if(i == 0 || i == 1 || i==8 || i==15 ){
        left= shift_left_once(left);
        right= shift_left_once(right);
    }
    // 3.2. For other rounds, the key_chunks
    // are shifted by two
    else{
        left= shift_left_twice(left);
        right= shift_left_twice(right);
    }
// 4. The chunks are combined
string combined_key = left + right;
string round_key = "";
// 5. Finally, the PC2 table is used to transpose
// the key bits
```

```
    for(int i = 0; i < 48; i++){
        round_key += combined_key[pc2[i]-1];
    }
    round_keys[i] = round_key;
        cout<<"Key "<<i+1<<": "<<round_keys[i]<<endl;
    }

}
int main(){
    string key = "1010101010111011000010010001100000100111001 1"
    "01101100110011011101";
    generate_keys(key);
}
```

Output



```
ktop\acads\7th sem\cnsl\"des
Key 1: 000110010100110011010000011100101101111010001100
Key 2: 010001010110100001011000000110101011110011001110
Key 3: 000001101110110110100100101011001111010110110101
Key 4: 110110100010110100000001100101011011011101100011
Key 5: 011010011010011000101001111111101100100100010011
Key 6: 110000001100101001000111010000111010001110101110
Key 7: 011100001000101011010010110111011011001111000000
Key 8: 001101001111100000010001011110000110001100110110 1
Key 9: 100001001011101101000100011100111101110011001100
Key 10: 000000100111011001010111000010001011010110111111
Key 11: 011011010101010101100000101011110111110010100101
Key 12: 110000101100000111101001011010100100101111110011
Key 13: 100110011100001100010011100101111100100100011111
Key 14: 001001010001101110001011110001110001011111010000
Key 15: 001100110011000011000101110110011010001101101101
Key 16: 000110000000111000101110101110101110001100110110 1
```

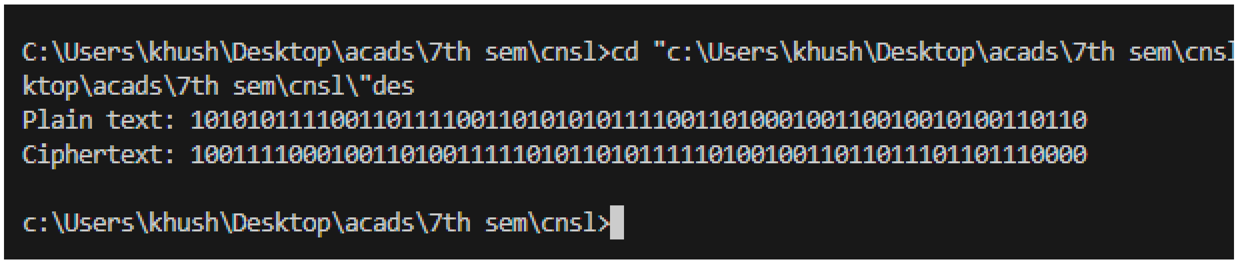
2. Encrypting plaintext to obtain ciphertext

```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;
// Array to hold 16 keys
string round_keys[16];
// String to hold the plain text
string pt;
// Function to convert a number in decimal to binary
string convertDecimalToBinary(int decimal)
{
    string binary;
    while(decimal != 0) {
        binary = (decimal % 2 == 0 ? "0" : "1") + binary;
        decimal = decimal/2;
    }
    while(binary.length() < 4){
        binary = "0" + binary;
    }
    return binary;
}
// Function to convert a number in binary to decimal
int convertBinaryToDecimal(string binary)
{
    int decimal = 0;
    int counter = 0;
    int size = binary.length();
    for(int i = size-1; i >= 0; i--)
    {
        if(binary[i] == '1'){
            decimal += pow(2, counter);
        }
    counter++;
    }
    return decimal;
}
// Function to do a circular left shift by 1
string shift_left_once(string key_chunk){
    string shifted="";
        for(int i = 1; i < 28; i++){
            shifted += key_chunk[i];
        }
        shifted += key_chunk[0];
    return shifted;
}
// Function to do a circular left shift by 2
string shift_left_twice(string key_chunk){
    string shifted="";
    for(int i = 0; i < 2; i++){
        for(int j = 1; j < 28; j++){
            shifted += key_chunk[j];
        }
        shifted += key_chunk[0];
        key_chunk= shifted;
        shifted ="";
    }
    return key_chunk;
}
// Function to compute xor between two strings
string Xor(string a, string b){
    string result = "";
    int size = b.size();
    for(int i = 0; i < size; i++){
        if(a[i] != b[i]){
            result += "1";
        }
        else{
            result += "0";
        }
    }
    return result;
}
// Function to generate the 16 keys.
void generate_keys(string key){
    // The PC1 table
    int pc1[56] = {
    57,49,41,33,25,17,9,
    1,58,50,42,34,26,18,
10,2,59,51,43,35,27,
19,11,3,60,52,44,36,
63,55,47,39,31,23,15,
7,62,54,46,38,30,22,
14,6,61,53,45,37,29,
21,13,5,28,20,12,4
};
// The PC2 table
int pc2[48] = {
14,17,11,24,1,5,
3,28,15,6,21,10,
23,19,12,4,26,8,
16,7,27,20,13,2,
41,52,31,37,47,55,
30,40,51,45,33,48,
44,49,39,56,34,53,
46,42,50,36,29,32
};
// 1. Compressing the key using the PC1 table
string perm_key ="";
for(int i = 0; i < 56; i++){
    perm_key+= key[pc1[i]-1];
}
// 2. Dividing the key into two equal halves
string left= perm_key.substr(0, 28);
string right= perm_key.substr(28, 28);
for(int i=0; i<16; i++){
    // 3.1. For rounds 1, 2, 9, 16 the key_chunks
    // are shifted by one.
    if(i == 0 || i == 1 || i==8 || i==15 ){
        left= shift_left_once(left);
        right= shift_left_once(right);
    }
    // 3.2. For other rounds, the key_chunks
    // are shifted by two
    else{
        left= shift_left_twice(left);
        right= shift_left_twice(right);
    }
    // Combining the two chunks
```

```
        string combined_key = left + right;
        string round_key = "";
        // Finally, using the PC2 table to transpose the key bits
        for(int i = 0; i < 48; i++){
            round_key += combined_key[pc2[i]-1];
        }
        round_keys[i] = round_key;
    }

}
// Implementing the algorithm
string DES(){
    // The initial permutation table
    int initial_permutation[64] = {
    58,50,42,34,26,18,10,2,
    60,52,44,36,28,20,12,4,
    62,54,46,38,30,22,14,6,
    64,56,48,40,32,24,16,8,
    57,49,41,33,25,17,9,1,
    59,51,43,35,27,19,11,3,
    61,53,45,37,29,21,13,5,
    63,55,47,39,31,23,15,7
    };
    // The expansion table
    int expansion_table[48] = {
    32,1,2,3,4,5,4,5,
    6,7,8,9,8,9,10,11,
    12,13,12,13,14,15,16,17,
    16,17,18,19,20,21,20,21,
    22,23,24,25,24,25,26,27,
    28,29,28,29,30,31,32,1
    };
    // The substitution boxes. The should contain values
    // from 0 to 15 in any order.
    int substition_boxes[8][4][16]=
    {{
        14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,
        0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,
        4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,
        15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13
    },
    {
        15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,
        3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,
        0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,
        13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9
    },
    {
        10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,
        13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,
        13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,
        1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12
    },
    {
        7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,
        13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,
        10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,
        3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14
    },
    {
        2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,
        14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,
        4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,
        11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3
    },
    {
        12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,
        10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,
        9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,
        4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13
    },
    {
        4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,
        13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,
        1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,
        6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12
    },
    {
        13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,
        1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,
        7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,
        2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11
    }};
    // The permutation table
    int permutation_tab[32] = {
    16,7,20,21,29,12,28,17,
    1,15,23,26,5,18,31,10,
    2,8,24,14,32,27,3,9,
    19,13,30,6,22,11,4,25
    };
    // The inverse permutation table
    int inverse_permutation[64]= {
    40,8,48,16,56,24,64,32,
    39,7,47,15,55,23,63,31,
    38,6,46,14,54,22,62,30,
    37,5,45,13,53,21,61,29,
    36,4,44,12,52,20,60,28,
    35,3,43,11,51,19,59,27,
    34,2,42,10,50,18,58,26,
    33,1,41,9,49,17,57,25
    };
    //1. Applying the initial permutation
    string perm = "";
    for(int i = 0; i < 64; i++){
        perm += pt[initial_permutation[i]-1];
    }
    // 2. Dividing the result into two equal halves
    string left = perm.substr(0, 32);
    string right = perm.substr(32, 32);
    // The plain text is encrypted 16 times
    for(int i=0; i<16; i++) {
        string right_expanded = "";
        // 3.1. The right half of the plain text is expanded
        for(int i = 0; i < 48; i++) {
            right_expanded += right[expansion_table[i]-1];
    };  // 3.3. The result is xored with a key
        string xored = Xor(round_keys[i], right_expanded);
        string res = "";
        // 3.4. The result is divided into 8 equal parts and passed
        // through 8 substitution boxes. After passing through a
```

```
        // substituion box, each box is reduces from 6 to 4 bits.
        for(int i=0;i<8; i++){
            // Finding row and column indices to lookup the
            // substituition box
            string row1= xored.substr(i*6,1) + xored.substr(i*6 + 5,1);
            int row = convertBinaryToDecimal(row1);
            string col1 = xored.substr(i*6 + 1,1) + xored.substr(i*6 +
2,1) + xored.substr(i*6 + 3,1) + xored.substr(i*6 + 4,1);;
            int col = convertBinaryToDecimal(col1);
            int val = substition_boxes[i][row][col];
            res += convertDecimalToBinary(val);
        }
        // 3.5. Another permutation is applied
        string perm2 ="";
        for(int i = 0; i < 32; i++){
            perm2 += res[permutation_tab[i]-1];
        }
        // 3.6. The result is xored with the left half
        xored = Xor(perm2, left);
        // 3.7. The left and the right parts of the plain text are swapped
        left = xored;
        if(i < 15){
            string temp = right;
            right = xored;
            left = temp;
        }
    }
    // 4. The halves of the plain text are applied
    string combined_text = left + right;
    string ciphertext ="";
    // The inverse of the initial permuttaion is applied
    for(int i = 0; i < 64; i++){
        ciphertext+= combined_text[inverse_permutation[i]-1];
    }
    //And we finally get the cipher text
    return ciphertext;
}
int main(){
    // A 64 bit key
    string key=
"1010101010111011000010010001100000100111001101101100110011011101";
    // A block of plain text of 64 bits
    pt=
"1010101111001101111001101010101111001101000100110010010100110110";
    // Calling the function to generate 16 keys
    generate_keys(key);
    cout<<"Plain text: "<<pt<<endl;
    // Applying the algo
    string ct= DES();
    cout<<"Ciphertext: "<<ct<<endl;
}
```

Output:



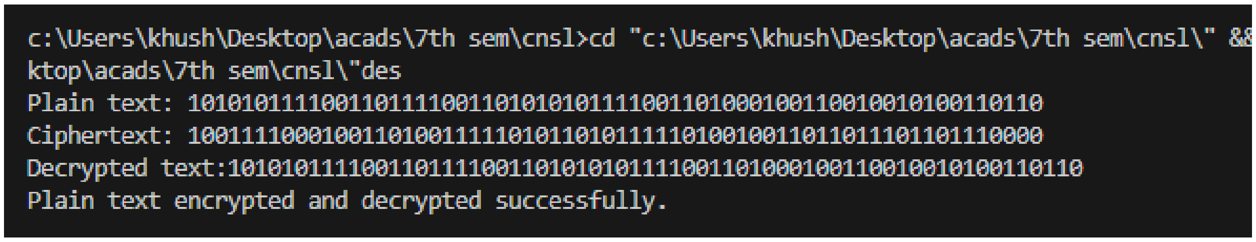
```
C:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Desktop\acads\7th sem\cnsl
ktop\acads\7th sem\cnsl\"des
Plain text: 1010101111001101111001101010101111001101000100110010010100110110
Ciphertext: 1001111000100110100111110101101011111010010011011011101101110000

c:\Users\khush\Desktop\acads\7th sem\cnsl>
```

3. Decrypting ciphertext to obtain plain text

```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;
// Array to hold 16 keys
string round_keys[16];
// String to hold the plain text
string pt;
// Function to convert a number in decimal to binary
string convertDecimalToBinary(int decimal)
{
    string binary;
    while(decimal != 0) {
        binary = (decimal % 2 == 0 ? "0" : "1") + binary;
        decimal = decimal/2;
    }
    while(binary.length() < 4){
        binary = "0" + binary;
    }
    return binary;
}
// Function to convert a number in binary to decimal
int convertBinaryToDecimal(string binary)
{
    int decimal = 0;
    int counter = 0;
    int size = binary.length();
    for(int i = size-1; i >= 0; i--)
    {
        if(binary[i] == '1'){
            decimal += pow(2, counter);
        }
    counter++;
    }
    return decimal;
}
// Function to do a circular left shift by 1
string shift_left_once(string key_chunk){
    string shifted="";
        for(int i = 1; i < 28; i++){
            shifted += key_chunk[i];
        }
        shifted += key_chunk[0];
    return shifted;
}
// Function to do a circular left shift by 2
string shift_left_twice(string key_chunk){
    string shifted="";
    for(int i = 0; i < 2; i++){
        for(int j = 1; j < 28; j++){
            shifted += key_chunk[j];
        }
        shifted += key_chunk[0];
        key_chunk= shifted;
        shifted ="";
    }
    return key_chunk;
}
// Function to compute xor between two strings
string Xor(string a, string b){
    string result = "";
    int size = b.size();
    for(int i = 0; i < size; i++){
        if(a[i] != b[i]){
            result += "1";
        }
        else{
            result += "0";
        }
    }
    return result;
}
// Function to generate the 16 keys.
void generate_keys(string key){
    // The PC1 table
    int pc1[56] = {
    57,49,41,33,25,17,9,
    1,58,50,42,34,26,18,
10,2,59,51,43,35,27,
19,11,3,60,52,44,36,
63,55,47,39,31,23,15,
7,62,54,46,38,30,22,
14,6,61,53,45,37,29,
21,13,5,28,20,12,4
};
// The PC2 table
int pc2[48] = {
14,17,11,24,1,5,
3,28,15,6,21,10,
23,19,12,4,26,8,
16,7,27,20,13,2,
41,52,31,37,47,55,
30,40,51,45,33,48,
44,49,39,56,34,53,
46,42,50,36,29,32
};
// 1. Compressing the key using the PC1 table
string perm_key ="";
for(int i = 0; i < 56; i++){
    perm_key+= key[pc1[i]-1];
}
// 2. Dividing the key into two equal halves
string left= perm_key.substr(0, 28);
string right= perm_key.substr(28, 28);
for(int i=0; i<16; i++){
    // 3.1. For rounds 1, 2, 9, 16 the key_chunks
    // are shifted by one.
    if(i == 0 || i == 1 || i==8 || i==15 ){
        left= shift_left_once(left);
        right= shift_left_once(right);
    }
    // 3.2. For other rounds, the key_chunks
    // are shifted by two
    else{
        left= shift_left_twice(left);
        right= shift_left_twice(right);
    }
    // Combining the two chunks
```

```
        string combined_key = left + right;
        string round_key = "";
        // Finally, using the PC2 table to transpose the key bits
        for(int i = 0; i < 48; i++){
            round_key += combined_key[pc2[i]-1];
        }
        round_keys[i] = round_key;
    }

}
// Implementing the algorithm
string DES(){
    // The initial permutation table
    int initial_permutation[64] = {
    58,50,42,34,26,18,10,2,
    60,52,44,36,28,20,12,4,
    62,54,46,38,30,22,14,6,
    64,56,48,40,32,24,16,8,
    57,49,41,33,25,17,9,1,
    59,51,43,35,27,19,11,3,
    61,53,45,37,29,21,13,5,
    63,55,47,39,31,23,15,7
    };
    // The expansion table
    int expansion_table[48] = {
    32,1,2,3,4,5,4,5,
    6,7,8,9,8,9,10,11,
    12,13,12,13,14,15,16,17,
    16,17,18,19,20,21,20,21,
    22,23,24,25,24,25,26,27,
    28,29,28,29,30,31,32,1
    };
    // The substitution boxes. The should contain values
    // from 0 to 15 in any order.
    int substition_boxes[8][4][16]=
    {{
        14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,
        0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,
        4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,
        15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13
    },
    {
        15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,
        3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,
        0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,
        13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9
    },
    {
        10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,
        13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,
        13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,
        1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12
    },
    {
        7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,
        13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,
        10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,
        3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14
    },
    {
        2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,
        14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,
        4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,
        11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3
    },
    {
        12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,
        10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,
        9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,
        4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13
    },
    {
        4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,
        13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,
        1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,
        6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12
    },
    {
        13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,
        1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,
        7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,
        2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11
    }};
    // The permutation table
    int permutation_tab[32] = {
    16,7,20,21,29,12,28,17,
    1,15,23,26,5,18,31,10,
    2,8,24,14,32,27,3,9,
    19,13,30,6,22,11,4,25
    };
    // The inverse permutation table
    int inverse_permutation[64]= {
    40,8,48,16,56,24,64,32,
    39,7,47,15,55,23,63,31,
    38,6,46,14,54,22,62,30,
    37,5,45,13,53,21,61,29,
    36,4,44,12,52,20,60,28,
    35,3,43,11,51,19,59,27,
    34,2,42,10,50,18,58,26,
    33,1,41,9,49,17,57,25
    };
    //1. Applying the initial permutation
    string perm = "";
    for(int i = 0; i < 64; i++){
        perm += pt[initial_permutation[i]-1];
    }
    // 2. Dividing the result into two equal halves
    string left = perm.substr(0, 32);
    string right = perm.substr(32, 32);
    // The plain text is encrypted 16 times
    for(int i=0; i<16; i++) {
        string right_expanded = "";
        // 3.1. The right half of the plain text is expanded
        for(int i = 0; i < 48; i++) {
            right_expanded += right[expansion_table[i]-1];
    };  // 3.3. The result is xored with a key
        string xored = Xor(round_keys[i], right_expanded);
        string res = "";
        // 3.4. The result is divided into 8 equal parts and passed
        // through 8 substitution boxes. After passing through a
        // substituion box, each box is reduces from 6 to 4 bits.
        for(int i=0;i<8; i++){
            // Finding row and column indices to lookup the
            // substituition box
            string row1= xored.substr(i*6,1) + xored.substr(i*6 + 5,1);
            int row = convertBinaryToDecimal(row1);
            string col1 = xored.substr(i*6 + 1,1) + xored.substr(i*6 +
2,1) + xored.substr(i*6 + 3,1) + xored.substr(i*6 + 4,1);;
            int col = convertBinaryToDecimal(col1);
            int val = substition_boxes[i][row][col];
            res += convertDecimalToBinary(val);
        }
        // 3.5. Another permutation is applied
        string perm2 ="";
        for(int i = 0; i < 32; i++){
            perm2 += res[permutation_tab[i]-1];
        }
        // 3.6. The result is xored with the left half
        xored = Xor(perm2, left);
        // 3.7. The left and the right parts of the plain text are swapped
        left = xored;
        if(i < 15){
            string temp = right;
            right = xored;
            left = temp;
        }
    }
    // 4. The halves of the plain text are applied
    string combined_text = left + right;
    string ciphertext ="";
    // The inverse of the initial permuttaion is applied
    for(int i = 0; i < 64; i++){
        ciphertext+= combined_text[inverse_permutation[i]-1];
    }
    //And we finally get the cipher text
    return ciphertext;
}
int main(){
    // A 64 bit key
    string key=
"1010101010111011000010010001100000100111001101101100110011011101";
    // A block of plain text of 64 bits
    pt=
"1010101111001101111001101010101111001101000100110010010100110110";
    string apt = pt;
    // Calling the function to generate 16 keys
    generate_keys(key);
    cout<<"Plain text: "<<pt<<endl;
    // Applying the algo
    string ct= DES();
    cout<<"Ciphertext: "<<ct<<endl;
    // Reversing the round_keys array for decryption
    int i = 15;
    int j = 0;
    while(i > j)
    {
        string temp = round_keys[i];
        round_keys[i] = round_keys[j];
        round_keys[j] = temp;
        i--;
        j++;
    }
    pt = ct;
    string decrypted = DES();
    cout<<"Decrypted text:"<<decrypted<<endl;
    // Comapring the initial plain text with the decrypted text
    if (decrypted == apt){
        cout<<"Plain text encrypted and decrypted successfully."<<endl;
    }
}
```

Output



```
c:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Desktop\acads\7th sem\cnsl\" &&
ktop\acads\7th sem\cnsl\"des
Plain text: 1010101111001101111001101010101111001101000100110010010100110110
Ciphertext: 1001111000100110100111110101101011111010010011011011101101110000
Decrypted text:1010101111001101111001101010101111001101000100110010010100110110
Plain text encrypted and decrypted successfully.
```