**IMPLEMENTATION OF CAESAR CIPHER**

**AIM:**

To implement the simple substitution technique named Caesar cipher using C language.

**DESCRIPTION:**

To encrypt a message with a Caesar cipher, each letter in the message is changed using a simple rule: shift by three. Each letter is replaced by the letter three letters ahead in the alphabet. A becomes D, B becomes E, and so on. For the last letters, think of the alphabet as a circle and "wrap around". W becomes Z, X becomes A, Y becomes B, and Z becomes C. To change a message back, each letter is replaced by the one three before it.

**ALGORITHM:**

**STEP-1:** Read the plain text from the user.

**STEP-2:** Read the key value from the user.

**STEP-3:** To encrypt the text by adding the key with eachcharacter in the plain text.

**STEP-4:** To decrypt the cipher subtract the key from each character in the cipher text.

**STEP-5:** Display the cipher text obtained above.

**STEP-6:** Display the decrypt text obtained above.

**PROGRAM: (Caesar Cipher)**

#include <iostream>

using namespace std;

int main()

{

string text;

cout<<"Enter text:";

cin>>text;

int s;

cout<<"Enter Shift:";

cin>>s;

string result = "",d="";

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

{

if (isupper(text[i]))

result += char(int(text[i]+s-65)%26 +65);

else

result += char(int(text[i]+s-97)%26 +97);

}

for(int i=0;i<result.length();i++)

{

if(isupper(result[i]))

d+=char(int(result[i]-s-65)%26 +65);

else

d+=char(int(result[i]-s-97)%26 + 97);

}

cout<<"Text : " << text;

cout<<"\nShift: " << s;

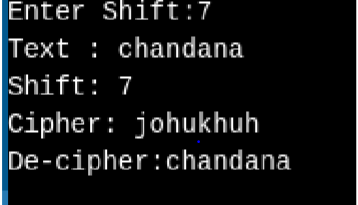
cout<<"\nCipher: " << result;

cout<<"\nDe-cipher:"<<d;

return 0;

}

**OUTPUT:**



**IMPLEMENTATION OF SUBSTITUTION CIPHER**

**AIM:**

To implement the substitution cipher using C language.

**DESCRIPTION:**

2.1) During encryption, replacement of plain text characters using a rotate of +13 characters at a time down the alphabet is done to generate cipher text and rotate of -13 characters up the alphabet is done to obtain original plain text. (standard method)

2.2) For each character in plain text, different units of characters can be replaced so that cipher text can be generated.

**ALGORITHM (2.1):**

**STEP-1:** Read the plain text from the user.

**STEP-2:** To encrypt the text by adding 13 with eachcharacter in the plain text.

**STEP-3:** To decrypt the cipher subtract 13 from each character in the cipher text.

**STEP-4:** Display the cipher text obtained above.

**STEP-5:** Display the decrypt text obtained above.

**ALGORITHM (2.2):**

**STEP-1:** Read the plain text from the user.

**STEP-2:** Initialize the shift value.

**STEP-3:** To encrypt the text by adding the shift value with eachcharacter in the plain text and for every iteration increment the shift value.

**STEP-4:** To decrypt the cipher subtract the shift value from each character in the cipher.

**STEP-5:** Display the cipher text obtained above.

**STEP-6:** Display the decrypt text obtained above.

**PROGRAM-1:**

#include <iostream>

using namespace std;

int main()

{

string text;

cout<<"Enter text:";

cin>>text;

string result = "",d="";

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

{

if (isupper(text[i]))

result += char(int(text[i]+13-65)%26 +65);

else

result += char(int(text[i]+13-97)%26 +97);

}

for(int i=0;i<result.length();i++)

{

if(isupper(result[i]))

d+=char(int(result[i]+26-13-65)%26 +65);

else

d+=char(int(result[i]+26-13-97)%26 + 97);

}

cout<<"Text : " << text;

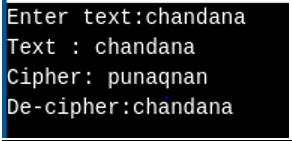
cout<<"\nCipher: " << result;

cout<<"\nDe-cipher:"<<d;

return 0;

}

**OUTPUT:**



**PROGRAM-2:**

#include <iostream>

using namespace std;

int main()

{

string text;

cout<<"Enter text:";

cin>>text;

int s=3;

string result = "",d="";

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

{

if (isupper(text[i]))

result += char(int(text[i]+s-65)%26 +65);

else

result += char(int(text[i]+s-97)%26 +97);

s=(s+i+1)%26;

}

s=3;

for(int i=0;i<result.length();i++)

{

if(isupper(result[i]))

d+=char(int(result[i]+26-s-65)%26 +65);

else

d+=char(int(result[i]+26-s-97)%26 + 97);

s=(s+i+1)%26;

}

cout<<"Text : " << text;

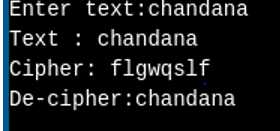
cout<<"\nCipher: " << result;

cout<<"\nDe-cipher:"<<d;

return 0;

}

**OUTPUT:**



**IMPLEMENTATION OF HILL CIPHER**

**AIM:**

To implement the hill cipher algorithm.

**DESCRIPTION:**

Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

**ALGORITHM :**

**STEP-1:** Read the plain text and key from the user.

**STEP-2:** Split the plain text into a matrix.

**STEP-3:** Arrange the keyword in a 2\*2 matrix.

**STEP-4:** Multiply the two matrices to obtain the cipher text.

**STEP-5:** Multiply the key inverse and cipher matrices to obtain the plain text.

**PROGRAM :**

#include<iostream>

#include<math.h>

using namespace std;

float encrypt[2][1], decrypt[2][1], a[2][2], b[2][2], mes[2][1], c[2][2];

void encryption();

void decryption();

void getKeyMessage();

void inverse();

int main() {

getKeyMessage();

encryption();

decryption();

}

void encryption() {

int i, j, k;

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

for(j = 0; j < 1; j++)

for(k = 0; k < 2; k++)

encrypt[i][j] = encrypt[i][j] + a[i][k] \* mes[k][j];

cout<<"\nEncrypted string is: ";

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

cout<<(char)(fmod(encrypt[i][0], 26) + 97);

}

void decryption() {

int i, j, k;

inverse();

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

for(j = 0; j < 1; j++)

for(k = 0; k < 2; k++)

decrypt[i][j] = decrypt[i][j] + b[i][k] \* encrypt[k][j];

cout<<"\nDecrypted string is: ";

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

cout<<(char)(fmod(decrypt[i][0], 26) + 97);

cout<<"\n";

}

void getKeyMessage() {

int i, j;

char msg[2];

cout<<"Enter 2x2 matrix for key (It should be inversible):\n";

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

for(j = 0; j < 2; j++) {

cin>>a[i][j];

c[i][j] = a[i][j];

}

cout<<"\nEnter a 2 letter string: ";

cin>>msg;

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

mes[i][0] = msg[i] - 97;

}

void inverse() {

int i, j, k;

float p, q;

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

for(j = 0; j < 2; j++) {

if(i == j)

b[i][j]=1;

else

b[i][j]=0;

}

for(k = 0; k < 2; k++) {

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

p = c[i][k];

q = c[k][k];

for(j = 0; j < 2; j++) {

if(i != k) {

c[i][j] = c[i][j]\*q - p\*c[k][j];

b[i][j] = b[i][j]\*q - p\*b[k][j];

}

}

}

}

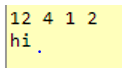
for(i = 0; i < 2; i++)

for(j = 0; j < 2; j++)

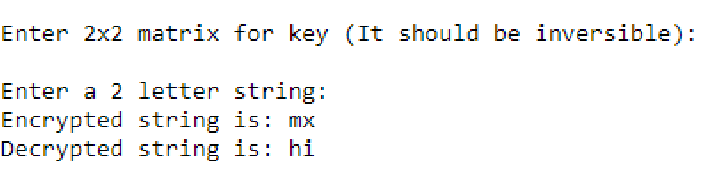
b[i][j] = b[i][j] / c[i][i];

}

**INPUT:**



**OUTPUT:**



**IMPLEMENTATION OF RSA**

**AIM:**

To implement RSA algorithm.

**DESCRIPTION:**

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone.

The public key is represented by the integers n and e; and, the private key, by the integer d. m represents the message. RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key.

**ALGORITHM :**

**STEP-1:** Select two co-prime numbers as p and q.

**STEP-2:** Compute n as the product of p and q.

**STEP-3:** Compute (p-1)\*(q-1) and store it in z.

**STEP-4:** Select a random prime number e that is less than that of z.

**STEP-5:** Compute the private key, d as e \* mod-1(z).

**STEP-6:** The cipher text is computed as messagee\* mod n.

**STEP-7:** Decryption is done as cipherdmod n.

**PROGRAM :**

#include<stdio.h>

#include<conio.h>

int phi,M,n,e,d,C,FLAG;

int check()

{

int i;

for(i=3;e%i==0 && phi%i==0;i+2)

{

FLAG = 1;

return 1;

}

FLAG = 0;

}

void encrypt()

{

int i;

C = 1;

for(i=0;i< e;i++)

C=C\*M%n;

C = C%n;

printf("\n\tEncrypted keyword : %d",C);

}

void decrypt(int c)

{

int i;

M = 1;

for(i=0;i< d;i++)

M=M\*c%n;

M = M%n;

printf("\n\tDecrypted keyword : %d",M);

}

int main()

{

int p,q,s;

printf("Enter Two Relatively Prime Numbers\t: ");

scanf("%d%d",&p,&q);

n = p\*q;

phi=(p-1)\*(q-1);

printf("\n\tF(n) phi value\t= %d",phi);

do

{

printf("\n\nEnter e which is prime number and less than phi \t: ",n);

scanf("%d",&e);

check();

}while(FLAG==1);

d = 1;

do

{

s = (d\*e)%phi;

d++;

}while(s!=1);

d = d-1;

printf("\n\tPublic Key\t: {%d,%d}",e,n);

printf("\n\tPrivate Key\t: {%d,%d}",d,n);

printf("\n\nEnter The Plain Text\t: ");

scanf("%d",&M);

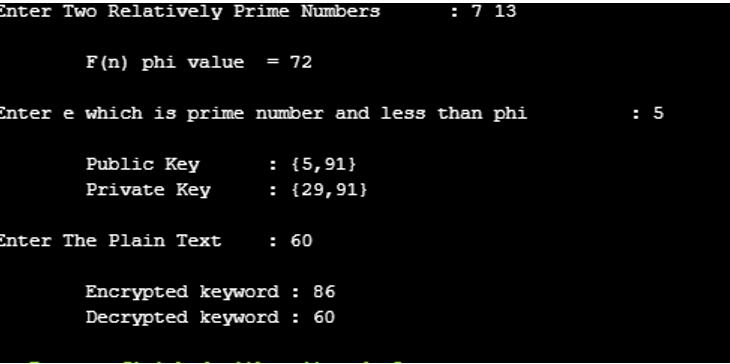
encrypt();

decrypt(C);

getch();

}

**OUTPUT:**



**IMPLEMENTATION OF DES**

**AIM:**

To write a program to implement Data Encryption Standard (DES).

**DESCRIPTION:**

DES is a symmetric encryption system that uses 64-bit blocks, 8 bits of which are used for parity checks. The key therefore has a "useful" length of 56 bits, which means that only 56 bits are actually used in the algorithm. The algorithm involves carrying out combinations, substitutions and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions. The key is ciphered on 64 bits and made of 16 blocks of 4 bits, generally denoted k1 to k16. Given that "only" 56 bits are actually used for encrypting, there can be 256 different keys.

**pyDES:**

This is a pure python implementation of the DES encryption algorithm. It is in pure python to avoid portability issues, since most DES implementations are programmed in C (for performance reasons).

Class initialization:-

pyDes.des(key, [mode], [IV], [pad], [padmode])

key -> Bytes containing the encryption key. 8 bytes for DES, 16 or 24 bytes for Triple DES

mode -> Optional argument for encryption type, can be either pyDes.ECB (Electronic Code Book) or pyDes.CBC (Cypher Block Chaining)

IV -> Optional Initial Value bytes, must be supplied if using CBC mode.

Length must be 8 bytes.

Pad -> Optional argument, set the pad character (PAD\_NORMAL) to use during all encrypt/decrpt operations done with this instance.

padmode -> Optional argument, set the padding mode (PAD\_NORMAL or PAD\_PKCS5) to use during all encrypt/decrpt operations done with this instance.

Common methods:-

encrypt(data, [pad], [padmode])

decrypt(data, [pad], [padmode])

data -> Bytes to be encrypted/decrypted

pad -> Optional argument. Only when using padmode of PAD\_NORMAL. For encryption, adds this characters to the end of the data block when data is not a multiple of 8 bytes. For decryption, will remove the trailing characters that match this pad character from the last 8 bytes of the unencrypted data block.

padmode -> Optional argument, set the padding mode, must be one of PAD\_NORMAL or PAD\_PKCS5). Defaults to PAD\_NORMAL.

**ALGORITHM:**

**STEP-1:** Read the 64-bit plain text.

**STEP-2:** Split it into two 32-bit blocks and store it in two different arrays.

**STEP-3:** Perform XOR operation between these two arrays.

**STEP-4:** The output obtained is stored as the second 32-bit sequence and the originalsecond 32-bit sequence forms the first part.

**STEP-5:** Thus the encrypted 64-bit cipher text is obtained in this way. Repeat the sameprocess for the remaining plain text characters.

**PROGRAM :**

from pyDes import des

key = "secret\_k"

text= "hey sri how r u?"

d = des(key)

ciphered = d.encrypt(text)

plain = d.decrypt(ciphered)

print ("Ciphered: ", ciphered)

print ("Deciphered: ", plain)

**OUTPUT:**

