EX. NO: 1(A)

IMPLEMENTATION OF CAESAR CIPHER

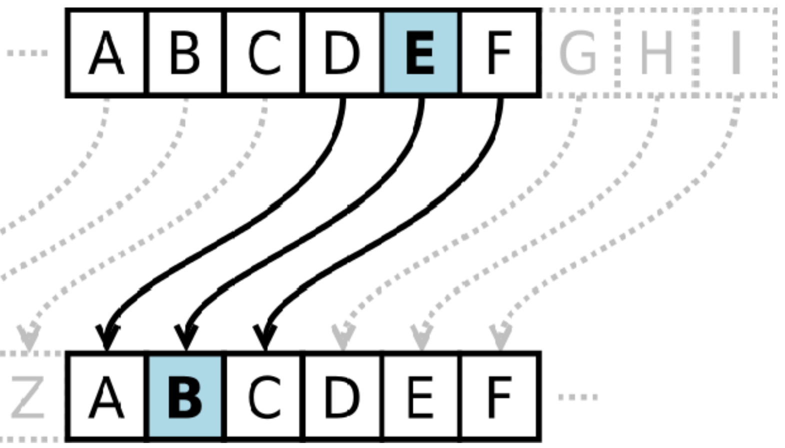
AIM:

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, we can 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.

EXAMPLE:



ALGORITHM:

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

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

STEP-3: If the key is positive then encrypt the text by adding the key with each character in the plain text.

STEP-4: Else subtract the key from the plain text.

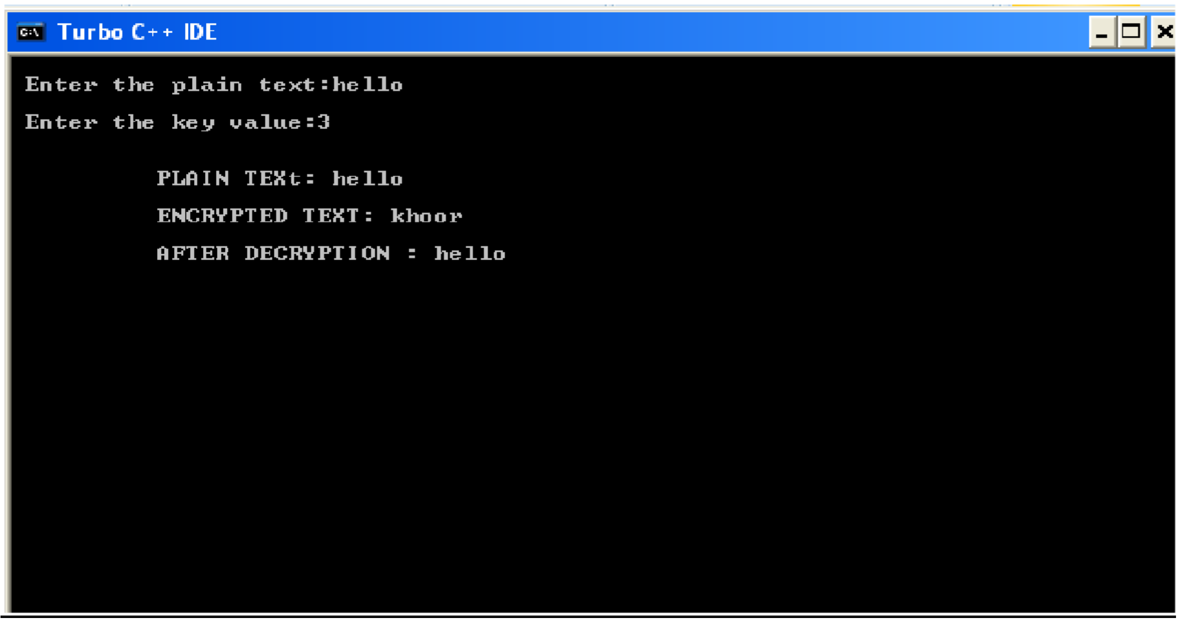
STEP-5: Display the cipher text obtained above.

PROGRAM: (Caesar Cipher)

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
#include <ctype.h>
void main()
```

```
{
 char plain[10], cipher[10];
 int key,i,length;
 int result;
 clrscr();
 printf("\n Enter the plain text:");
 scanf("%s", plain);
 printf("\n Enter the key value:");
 scanf("%d", &key);
 printf("\n \n \t  PLAIN TEXt: %s",plain);
 printf("\n \n \t  ENCRYPTED TEXT: ");
 for(i = 0, length = strlen(plain); i < length;  i++)
      {
      cipher[i]=plain[i] + key;
      if (isupper(plain[i]) && (cipher[i] > 'Z'))
      cipher[i] = cipher[i] - 26;
      if (islower(plain[i]) && (cipher[i] > 'z'))
      cipher[i] = cipher[i] - 26;
      printf("%c", cipher[i]);
      }
      printf("\n \n \t  AFTER DECRYPTION : ");
 for(i=0;i<length;i++)
      {
       plain[i]=cipher[i]-key;
      if(isupper(cipher[i])&&(plain[i]<'A'))
      plain[i]=plain[i]+26;
      if(islower(cipher[i])&&(plain[i]<'a'))
      plain[i]=plain[i]+26;
      printf("%c",plain[i]);
 }
getch();
}
```

<u>OUTPUT:</u>



```
Enter the plain text:hello
Enter the key value:3

         PLAIN TEXt: hello
         ENCRYPTED TEXT: khoor
         AFTER DECRYPTION : hello
```

IMPLEMENTATION OF PLAYFAIR CIPHER

AIM:

To write a C program to implement the Playfair Substitution technique.

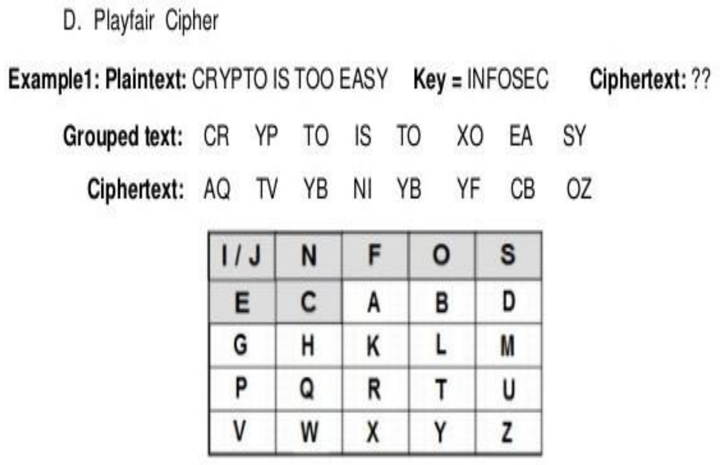
DESCRIPTION:

The Playfair cipher starts with creating a key table. The key table is a 5×5 grid of letters that will act as the key for encrypting your plaintext. Each of the 25 letters must be unique and one letter of the alphabet is omitted from the table (as there are 25 spots and 26 letters in the alphabet).

To encrypt a message, one would break the message into digrams (groups of 2 letters) such that, for example, "HelloWorld" becomes "HE LL OW OR LD", and map them out on the key table. The two letters of the diagram are considered as the opposite corners of a rectangle in the key table. Note the relative position of the corners of this rectangle. Then apply the following 4 rules, in order, to each pair of letters in the plaintext:

1. If both letters are the same (or only one letter is left), add an "X" after the first letter
2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively
3. If the letters appear on the same column of your table, replace them with the letters immediately below respectively
4. If the letters are not on the same row or column, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair.

EXAMPLE:



D. Playfair Cipher

Example1: Plaintext: CRYPTO IS TOO EASY **Key =** INFOSEC **Ciphertext:** ??

Grouped text: CR YP TO IS TO XO EA SY

Ciphertext: AQ TV YB NI YB YF CB OZ

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

ALGORITHM:

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

STEP-2: Read the keyword from the user.

STEP-3: Arrange the keyword without duplicates in a 5*5 matrix in the row order and fill the remaining cells with missed out letters in alphabetical order. Note that 'i' and 'j' takes the same cell.

STEP-4: Group the plain text in pairs and match the corresponding corner letters by forming a rectangular grid.

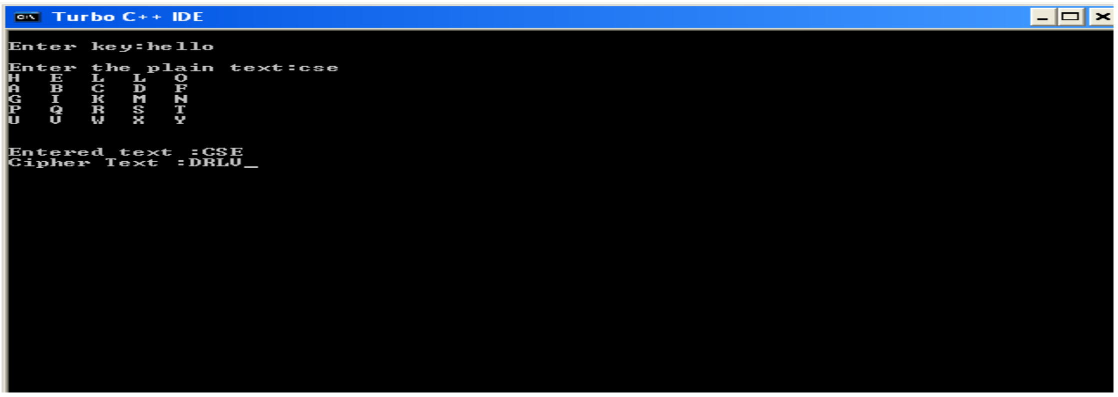
STEP-5: Display the obtained cipher text.

PROGRAM: (Playfair Cipher)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#define MX 5
void playfair(char ch1,char ch2, char key[MX][MX])
{
     int i,j,w,x,y,z;
     FILE *out;
if((out=fopen("cipher.txt","a+"))==NULL)
{
     printf("File Currupted.");
}
for(i=0;i<MX;i++)
{
     for(j=0;j<MX;j++)
     {
     if(ch1==key[i][j])
     {
     w=i;
     x=j;
     }
     else if(ch2==key[i][j])
     {
     y=i;
     z=j;
}}}
//printf("%d%d %d%d",w,x,y,z);
if(w==y)
{
     x=(x+1)%5;z=(z+1)%5;
     printf("%c%c",key[w][x],key[y][z]);
     fprintf(out, "%c%c",key[w][x],key[y][z]);
}
else if(x==z)
{
     w=(w+1)%5;y=(y+1)%5;
     printf("%c%c",key[w][x],key[y][z]);
     fprintf(out, "%c%c",key[w][x],key[y][z]);
 }
 else
 {
     printf("%c%c",key[w][z],key[y][x]);
     fprintf(out, "%c%c",key[w][z],key[y][x]);
 }
fclose(out);
}
void main()
{
     int i,j,k=0,l,m=0,n;
     char  key[MX][MX],keyminus[25],keystr[10],str[25]={0};
     char
     alpa[26]={'A','B','C','D','E','F','G','H','I','J','K','L'
     ,'M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'}
     ;
     clrscr();
     printf("\nEnter key:");
     gets(keystr);
     printf("\nEnter the plain text:");
     gets(str);
     n=strlen(keystr);
     //convert the characters to uppertext
     for (i=0; i<n; i++)
     {
          if(keystr[i]=='j')keystr[i]='i';
          else if(keystr[i]=='J')keystr[i]='I';
          keystr[i] = toupper(keystr[i]);
     }
     //convert all the characters  of plaintext  to uppertext
     for (i=0; i<strlen(str); i++)
     {
          if(str[i]=='j')str[i]='i';
          else if(str[i]=='J')str[i]='I';
          str[i] = toupper(str[i]);
     }
     j=0;
     for(i=0;i<26;i++)
     {
          for(k=0;k<n;k++)
          {
          if(keystr[k]==alpa[i])
          break;
          else if(alpa[i]=='J')
          break;
          }
          if(k==n)
          {
          keyminus[j]=alpa[i];j++;
          }
     }
    //construct key keymatrix
    k=0;
    for(i=0;i<MX;i++)
    {
         for(j=0;j<MX;j++)
         {
         if(k<n)
         {
         key[i][j]=keystr[k];
         k++;}
         else
         {
         key[i][j]=keyminus[m];m++;
         }
         printf("%c  ",key[i][j]);
         }
         printf("\n");
    }
    printf("\n\nEntered text :%s\nCipher Text :",str);
    for(i=0;i<strlen(str);i++)
    {
    if(str[i]=='J')str[i]='I';
    if(str[i+1]=='\0')
    playfair(str[i],'X',key);
    else
    {
              if(str[i+1]=='J')str[i+1]='I'; if(str[i]==str[i+1])
              playfair(str[i],'X',key);
         else
         {
         playfair(str[i],str[i+1],key);i++;
         }}

    }
getch();
}
```

AIM:



```
Enter key:hello

Enter the plain text:cse
H   E   L   L   O
A   B   C   D   F
G   I   K   M   N
P   Q   R   S   T
U   V   W   X   Y

Entered text :CSE
Cipher Text :DRLV_
```

IMPLEMENTATION OF HILL CIPHER

To write a C program to implement the hill cipher substitution techniques.

DESCRIPTION:

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 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. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible *n* × *n* matrices (modulo 26).

EXAMPLE:

$$\begin{bmatrix} 2 & 4 & 5 \\ 9 & 2 & 1 \\ 3 & 17 & 7 \end{bmatrix} \begin{bmatrix} 0 \\ 19 \\ 19 \end{bmatrix} = \begin{bmatrix} 171 \\ 57 \\ 456 \end{bmatrix} \pmod{26} = \begin{bmatrix} 15 \\ 5 \\ 14 \end{bmatrix} = \text{'PFO'}$$

ALGORITHM:

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

STEP-2: Split the plain text into groups of length three.

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

STEP-4: Multiply the two matrices to obtain the cipher text of length three.

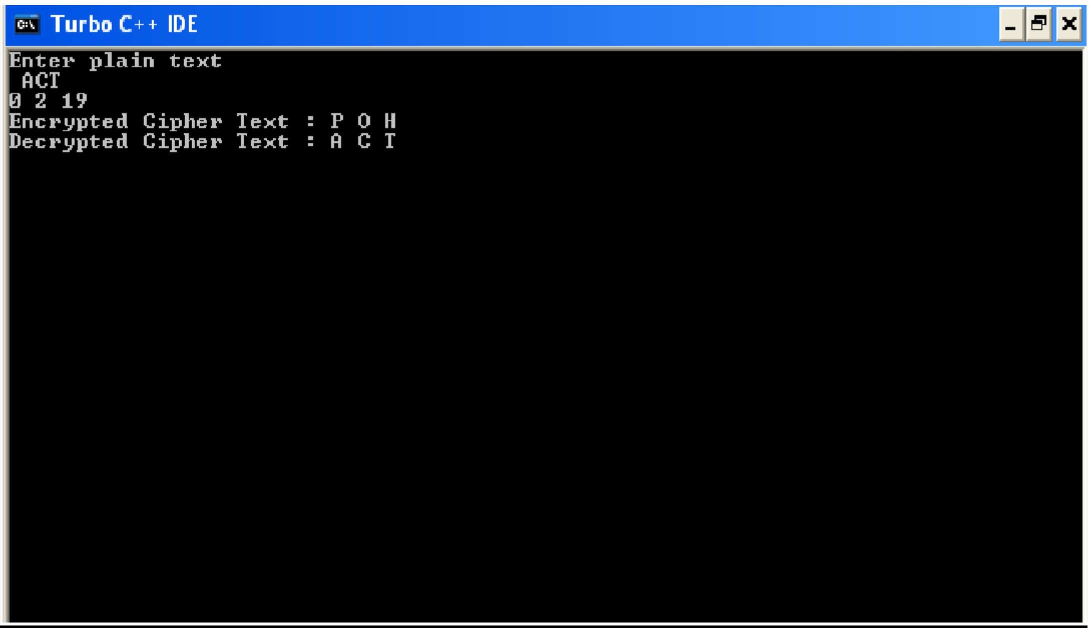
STEP-5: Combine all these groups to get the complete cipher text.

PROGRAM: (Hill Cipher)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main(){
     unsigned int a[3][3]={{6,24,1},{13,16,10},{20,17,15}};
     unsigned int b[3][3]={{8,5,10},{21,8,21},{21,12,8}};
     int i,j, t=0;
     unsigned int c[20],d[20];
     char msg[20];
     clrscr();
     printf("Enter plain text\n ");
     scanf("%s",msg);
     for(i=0;i<strlen(msg);i++)
     {    c[i]=msg[i]-65;
```

```
                printf("%d ",c[i]);
        }
        for(i=0;i<3;i++)
        {    t=0;
        for(j=0;j<3;j++)
        {
                t=t+(a[i][j]*c[j]);
        }
                d[i]=t%26;
        }
printf("\nEncrypted Cipher Text :");
for(i=0;i<3;i++)
printf(" %c",d[i]+65);
for(i=0;i<3;i++)
{
        t=0;
        for(j=0;j<3;j++)
        {
                t=t+(b[i][j]*d[j]);
        }
        c[i]=t%26;
}
printf("\nDecrypted Cipher Text :");
for(i=0;i<3;i++)
printf(" %c",c[i]+65);
getch();
return 0;
}
```

OUTPUT:



```
Enter plain text
 ACT
0 2 19
Encrypted Cipher Text : P O H
Decrypted Cipher Text : A C T
```

EX. NO: 1(D)

IMPLEME TATION OF VIGENERE CIPHER

AIM:

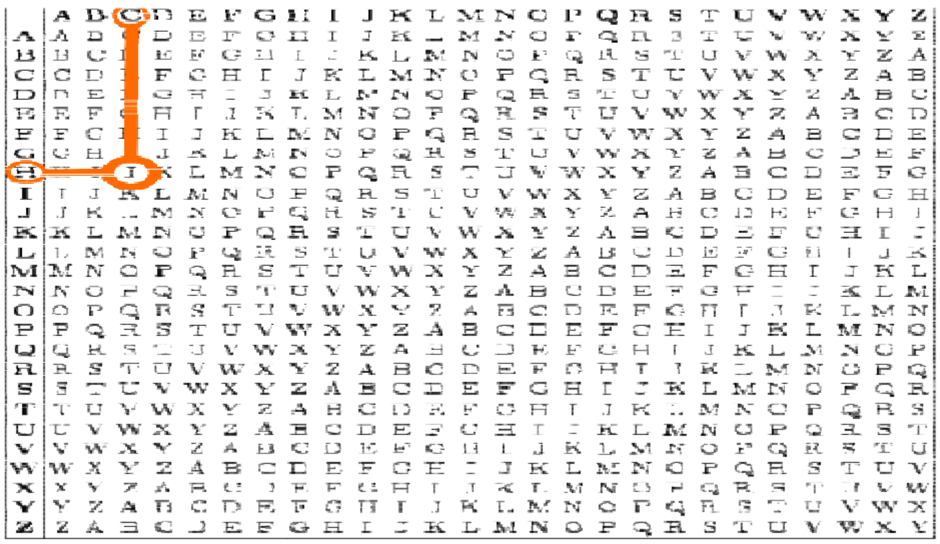
To implement the Vigenere Cipher substitution technique using C program.

DESCRIPTION:

To encrypt, a table of alphabets can be used, termed a tabula recta, Vigenère square, or Vigenère table. It consists of the alphabet written out 26 times in differ nt rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar ciphers. At different points in the encryption process, the cipher uses a different alphabet from one of the rows. The alphabet used at each point depends on a repeating keyword.

Each row starts with a key letter. The remainder of the row holds the letters A to Z. Although there are 26 key rows shown, you will only use as many keys as there are unique letters in the key string, here just 5 keys, {L, E, M, O, N}. For successive letters of the message, we are going to take successive letters of the key string, and encipher each message letter using its corresponding key row. Choose the next letter of the key, go along that row to find the column heading that matches the message character; the letter at the intersection of [key-row, msg-col] is the enciphered letter.

EXAMPLE:



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

ALGORITHM:

STEP-1: Arrange the alphabets in row and column of a 26*26 matrix.

STEP-2: Circulate the alphabets in each row to position left such that the first letter is attached to last.

STEP-3: Repeat this process for all 26 rows and construct the final key matrix.

STEP-4: The keyword and the plain text is read from the user.

STEP-5: The characters in the keyword are repeated sequentially so as to match with that of the plain text.

STEP-6: Pick the first letter of the plain text and that of the keyword as the row indices and column indices respectively.

STEP-7: The junction character where these two meet forms the cipher character.

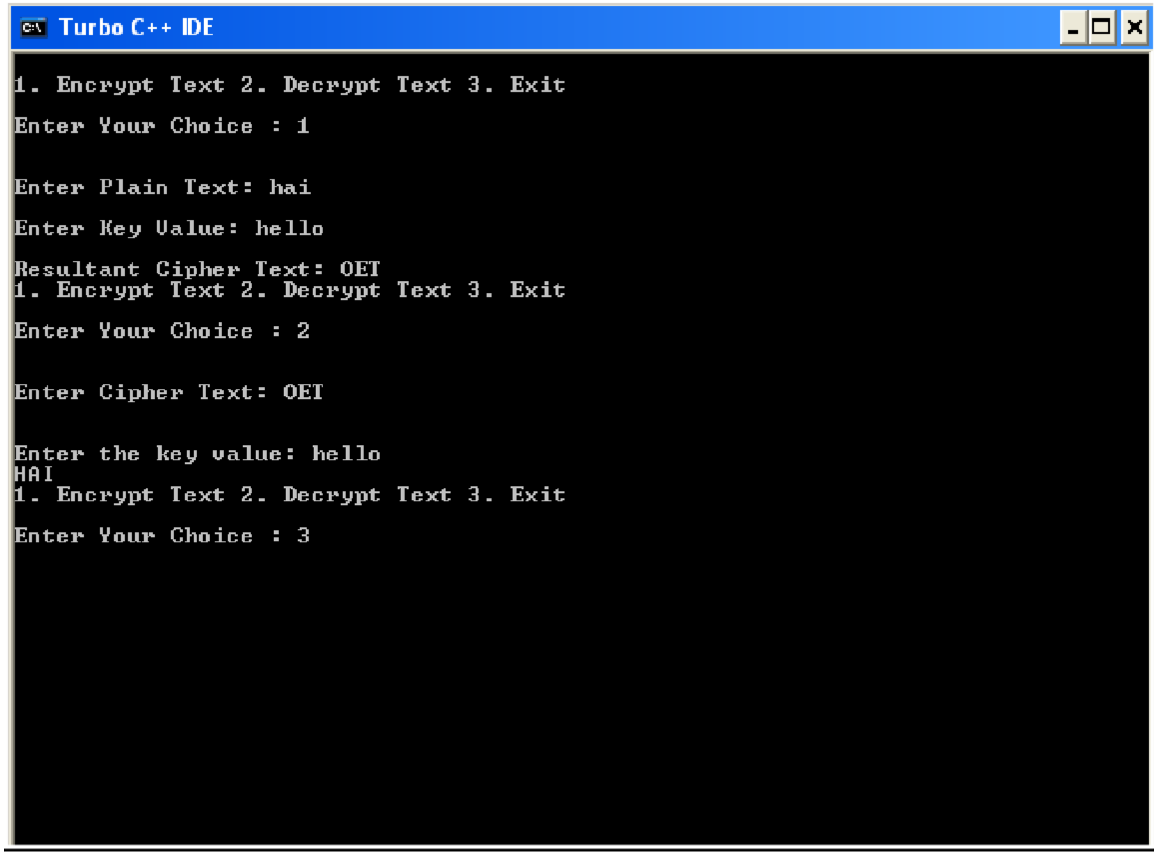
STEP-8: Repeat the above steps to generate the entire cipher text.

PROGRAM: (Vigenere Cipher)

```
#include <stdio.h>
#include<conio.h>
#include <ctype.h>
#include <string.h>
void encipher();
void decipher();
void main()
{
     int choice;
     clrscr();
     while(1)
     {
          printf("\n1. Encrypt Text");
          printf("\t2. Decrypt Text");
          printf("\t3. Exit");
          printf("\n\nEnter Your Choice : ");
          scanf("%d",&choice);
          if(choice == 3)
          exit(0);
          else if(choice == 1)
          encipher();
          else if(choice == 2)
          decipher();
          else
          printf("Please Enter Valid Option.");
     }
}
void encipher()
{
     unsigned int i,j;
     char input[50],key[10];
     printf("\n\nEnter Plain Text:  ");
```

```
        scanf("%s",input);
        printf("\nEnter Key Value: ");
        scanf("%s",key);
        printf("\nResultant Cipher Text: ");
        for(i=0,j=0;i<strlen(input);i++,j++)
        {
                if(j>=strlen(key))
                        {     j=0;
                        }
        printf("%c",65+(((toupper(input[i])-65)+(toupper(key[j])-
        65))%26));
        }}
void decipher()
{
        unsigned int i,j;
        char input[50],key[10];
        int value;
        printf("\n\nEnter Cipher Text: ");
        scanf("%s",input);
        printf("\n\nEnter the key value: ");
        scanf("%s",key);
for(i=0,j=0;i<strlen(input);i++,j++)
{
        if(j>=strlen(key))
        {     j=0;        }
        value = (toupper(input[i])-64)-(toupper(key[j])-64);
        if( value < 0)
        {  value = value * -1;
        }
        printf("%c",65 + (value % 26));
}}
```

OUTPUT:



```
1. Encrypt Text 2. Decrypt Text 3. Exit

Enter Your Choice : 1


Enter Plain Text: hai

Enter Key Value: hello

Resultant Cipher Text: OET
1. Encrypt Text 2. Decrypt Text 3. Exit

Enter Your Choice : 2


Enter Cipher Text: OET


Enter the key value: hello
HAI
1. Encrypt Text 2. Decrypt Text 3. Exit

Enter Your Choice : 3
```

EX. NO: 1(E)

IMPLEMENTATION OF RAIL FENCE – ROW & COLUMN TRANSFORMATION TECHNIQUE

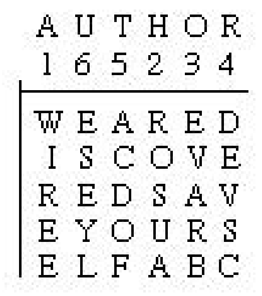
AIM:

To write a C program to implement the rail fence transposition technique.

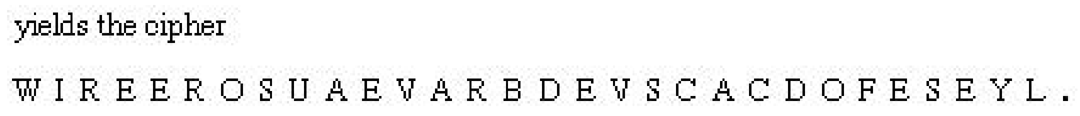
DESCRIPTION:

In the rail fence cipher, the plain text is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail. When we reach the top rail, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows.

EXAMPLE:



A	U	T	H	O	R
1	6	5	2	3	4
W	E	A	R	E	D
I	S	C	O	V	E
R	E	D	S	A	V
E	Y	O	U	R	S
E	L	F	A	B	C



yields the cipher

W I R E E R O S U A E V A R B D E V S C A C D O F E S E Y L .

ALGORITHM:

STEP-1: Read the Plain text.

STEP-2: Arrange the plain text in row columnar matrix format.

STEP-3: Now read the keyword depending on the number of columns of the plain text.

STEP-4: Arrange the characters of the keyword in sorted order and the corresponding columns of the plain text.

STEP-5: Read the characters row wise or column wise in the former order to get the cipher text.

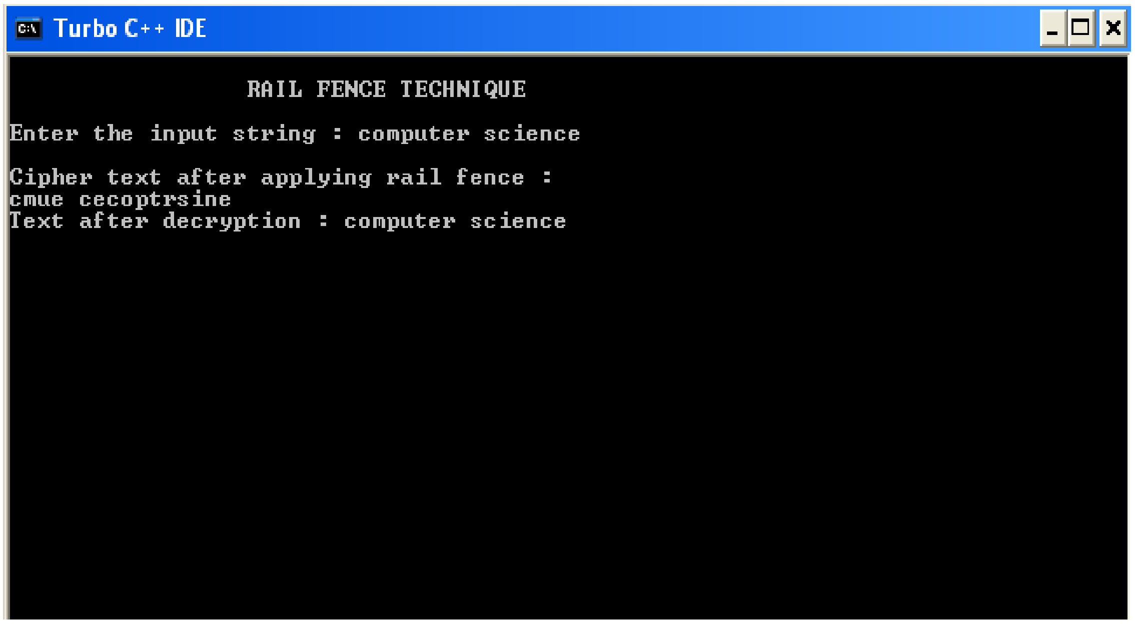
PROGRAM: (Rail Fence)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
     int i,j,k,l;
     char a[20],c[20],d[20];
     clrscr();
     printf("\n\t\t RAIL FENCE TECHNIQUE");
     printf("\n\nEnter the input string : ");
     gets(a);
     l=strlen(a);

/*Ciphering*/
for(i=0,j=0;i<l;i++)
{
     if(i%2==0)
     c[j++]=a[i];
}
for(i=0;i<l;i++)
{
     if(i%2==1)
     c[j++]=a[i];
}
c[j]='\0';
printf("\nCipher text after applying rail fence :");
printf("\n%s",c);

/*Deciphering*/
if(l%2==0)
     k=l/2;
else
     k=(l/2)+1;
for(i=0,j=0;i<k;i++)
{
    d[j]=c[i];
    j=j+2;
}
for(i=k,j=1;i<l;i++)
{
    d[j]=c[i];
    j=j+2;
}
d[l]='\0';
printf("\nText after decryption : ");
printf("%s",d);
getch();
}
```

OUTPUT:



```
                 RAIL FENCE TECHNIQUE

Enter the input string : computer science

Cipher text after applying rail fence :
cmue cecoptrsine
Text after decryption : computer science
```

EX. NO: 2(A)

IMPLEMENTATION OF DES

AIM:

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

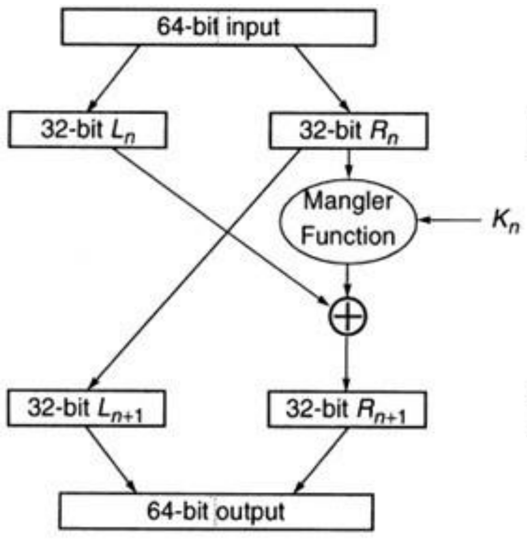
DESCRIPTION:

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 k_1 to k_{16}. Given that "only" 56 bits are actually used for encrypting, there can be 2^{56} different keys.

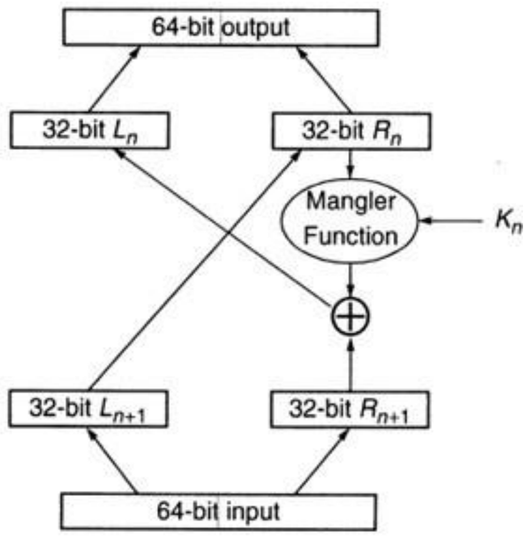
The main parts of the algorithm are as follows:

- ➢ Fractioning of the text into 64-bit blocks
- ➢ Initial permutation of blocks
- ➢ Breakdown of the blocks into two parts: left and right, named L and R
- ➢ Permutation and substitution steps repeated 16 times
- ➢ Re-joining of the left and right parts then inverse initial permutation

EXAMPLE:



Encryption



Decryption

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 original second 32-bit sequence forms the first part.

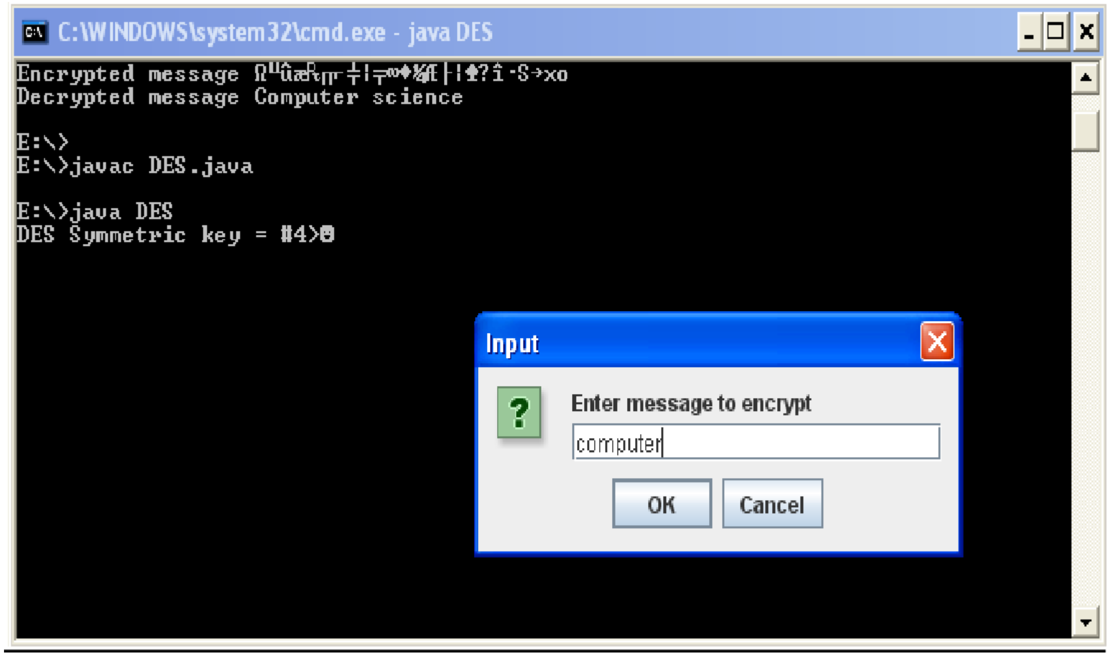
STEP-5: Thus the encrypted 64-bit cipher text is obtained in this way. Repeat the same process for the remaining plain text characters.

PROGRAM:

DES.java

```
import javax.swing.*;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Random ;
class DES {
     byte[] skey = new byte[1000];
     String skeyString;
     static byte[]  raw;
     String inputMessage,encryptedData,decryptedMessage;
public DES()
{
try
{
     generateSymmetricKey();
     inputMessage=JOptionPane.showInputDialog(null,"Enter
     message to encrypt");
     byte[] ibyte = inputMessage.getBytes();
     byte[] ebyte=encrypt(raw, ibyte);
     String encryptedData = new String(ebyte);
     System.out.println("Encrypted message "+encryptedData);
     JOptionPane.showMessageDialog(null,"Encrypted Data
     "+"\n"+encryptedData);
     byte[] dbyte= decrypt(raw,ebyte);
     String decryptedMessage = new String(dbyte);
     System.out.println("Decrypted message
     "+decryptedMessage);
     JOptionPane.showMessageDialog(null,"Decrypted Data
     "+"\n"+decryptedMessage);
}
catch(Exception e)
{
     System.out.println(e);
}
}
void generateSymmetricKey() {
try {
     Random r = new Random();
     int num =  r.nextInt(10000);
     String knum = String.valueOf(num);
     byte[] knumb = knum.getBytes();
     skey=getRawKey(knumb);
     skeyString = new String(skey);
     System.out.println("DES Symmetric key =  "+skeyString);
}
catch(Exception e)
{
     System.out.println(e);
}
}
private static byte[] getRawKey(byte[] seed) throws  Exception
{
     KeyGenerator kgen = KeyGenerator.getInstance("DES");
     SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
     sr.setSeed(seed);
     kgen.init(56, sr);
     SecretKey skey = kgen.generateKey();
     raw = skey.getEncoded();
     return raw;
}
private static byte[] encrypt(byte[] raw, byte[] clear) throws
Exception {
          SecretKeySpec skeySpec = new SecretKeySpec(raw,
          "DES");
          Cipher cipher = Cipher.getInstance("DES");
          cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
          byte[] encrypted = cipher.doFinal(clear);
          return encrypted;
}
private static byte[] decrypt(byte[] raw, byte[] encrypted)
throws Exception
{
          SecretKeySpec skeySpec = new SecretKeySpec(raw,
          "DES");
          Cipher cipher = Cipher.getInstance("DES");
          cipher.init(Cipher.DECRYPT_MODE, skeySpec);
          byte[] decrypted = cipher.doFinal(encrypted);
          return decrypted;
}
public static void main(String args[]) {
     DES des = new DES();
}
}
```

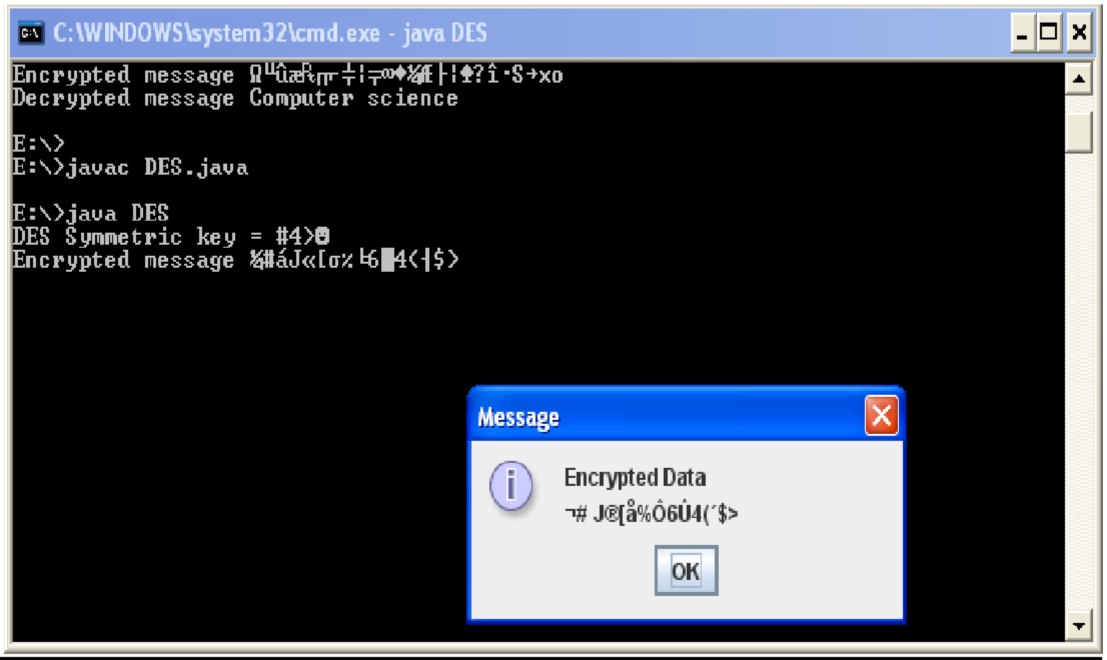
OUTPUT:



```
Encrypted message Ω╙ûæ₧╓┌╪¦╤∞◆¼Æ├¦♠?î·S→xo
Decrypted message Computer science

E:\>
E:\>javac DES.java

E:\>java DES
DES Symmetric key = #4>☻
```

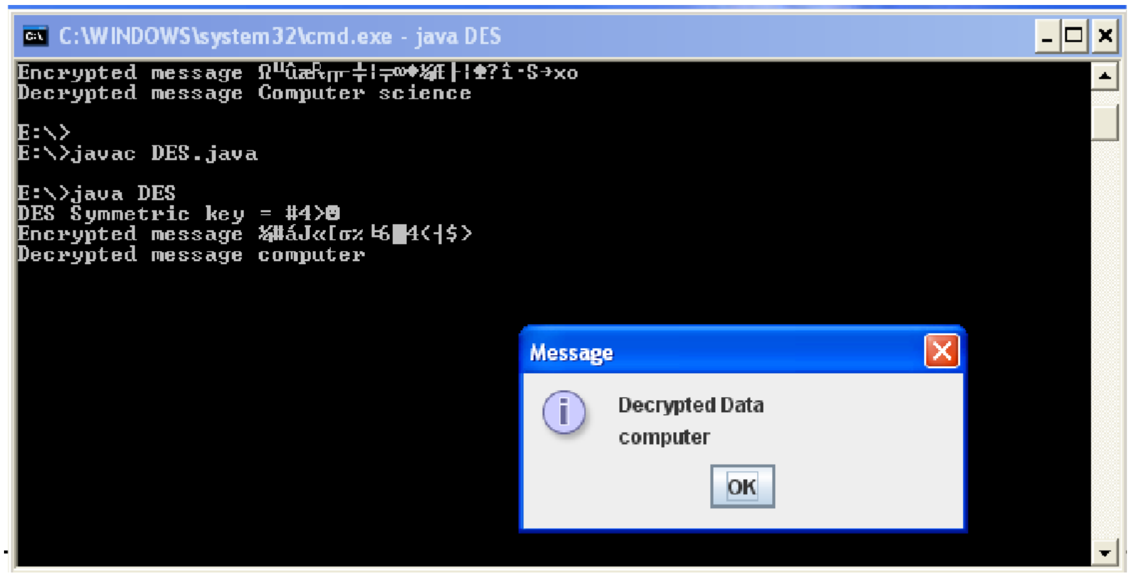


```
Encrypted message Ω╙ûæ₧╓┌╪¦╤∞◆¼Æ├¦♠?î·S→xo
Decrypted message Computer science

E:\>
E:\>javac DES.java

E:\>java DES
DES Symmetric key = #4>☻
Encrypted message ¼#áJ«[σ%╘6█4(┤$>
```

¬# J®[å%Ô6Û4(´$>



```
Encrypted message Ω╙ûæ₧╓╓─╪¦╤∞♦¼£├¦♠?î·S→xo
Decrypted message Computer science

E:\>
E:\>javac DES.java

E:\>java DES
DES Symmetric key = #4>☻
Encrypted message ¼#áJ«[σ% ╘6█4<┤$>
Decrypted message computer
```

Decrypted Data
computer

EX. NO: 2(B)

IMPLEMENTATION OF RSA

AIM:

To write a C program to implement the RSA encryption algorithm.

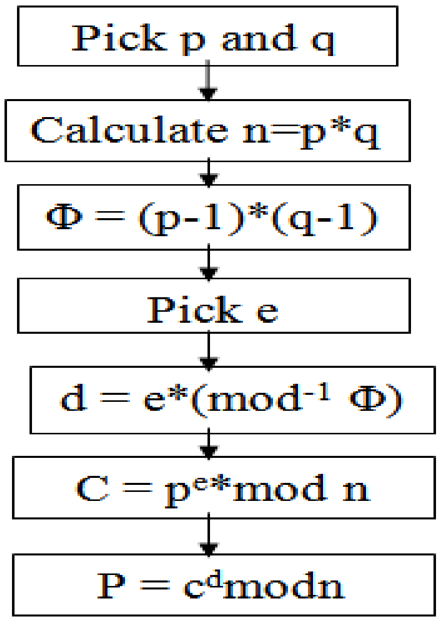
DESCRIPTION:

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. A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d and n such that with modular exponentiation for all integer m:

$$(m^e)^d = m \pmod{n}$$

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.

EXAMPLE:



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 message^{e} * mod n. **STEP-**

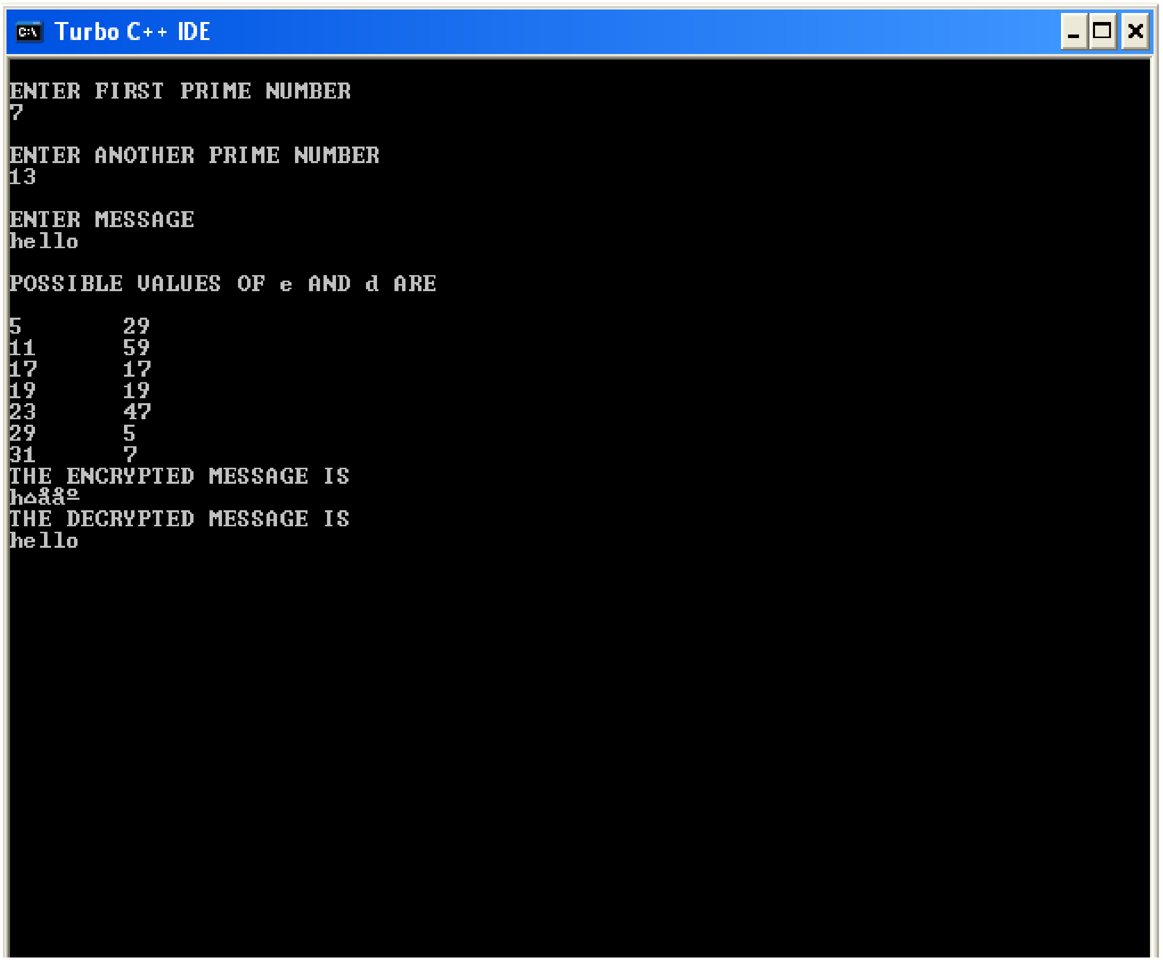
7: Decryption is done as cipher^{d}mod n.

PROGRAM: (RSA)

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
     long int
     p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;
     char msg[100];
     int prime(long int);
     void ce();
     long int cd(long int);
     void encrypt();
     void decrypt();
void main()
{
     clrscr();
     printf("\nENTER FIRST PRIME NUMBER\n");
     scanf("%d",&p);
     flag=prime(p);
     if(flag==0)
      {
          printf("\nWRONG INPUT\n");
          getch();
      }
      printf("\nENTER ANOTHER PRIME NUMBER\n");
      scanf("%d",&q);
      flag=prime(q);
      if(flag==0||p==q)
      {
       printf("\nWRONG INPUT\n");
       getch();
      }
      printf("\nENTER MESSAGE\n");
      fflush(stdin);
      scanf("%s",msg);
      for(i=0;msg[i]!=NULL;i++)
      m[i]=msg[i];
      n=p*q;
      t=(p-1)*(q-1);
      ce();
      printf("\nPOSSIBLE VALUES OF e AND d ARE\n");
      for(i=0;i<j-1;i++)
      printf("\n%ld\t%ld",e[i],d[i]);
      encrypt();
      decrypt();
      getch();
}
int prime(long int pr)
{
int i;
j=sqrt(pr);
for(i=2;i<=j;i++)
{
if(pr%i==0)
return 0;
}
return 1;
}
void ce()
{
int k;
k=0;
for(i=2;i<t;i++)
{
 if(t%i==0)
 continue;
 flag=prime(i);
 if(flag==1&&i!=p&&i!=q)
 {
 e[k]=i;
 flag=cd(e[k]);
 if(flag>0)
 {
 d[k]=flag;
 k++;
 }
 if(k==99)
 break;
 } } }
 long int cd(long int x)
 {
 long int k=1;
 while(1)
 {
 k=k+t;
 if(k%x==0)
 return(k/x);
 } }
 void encrypt() {
 long int pt,ct,key=e[0],k,len;
 i=0;
 len=strlen(msg);
```

```
while(i!=len) {
pt=m[i];
pt=pt-96;
k=1;
for(j=0;j<key;j++)
{  k=k*pt;
k=k%n;
}
temp[i]=k;
ct=k+96;
en[i]=ct;
i++;
}
en[i]=-1;
printf("\nTHE ENCRYPTED MESSAGE IS\n");
for(i=0;en[i]!=-1;i++)
printf("%c",en[i]);
}
void decrypt()
{
long int pt,ct,key=d[0],k;
i=0;
while(en[i]!=-1)
{
ct=temp[i];
k=1;
for(j=0;j<key;j++)
{
k=k*ct;
k=k%n;
}
pt=k+96;
m[i]=pt;
i++;
}
m[i]=-1;
printf("\nTHE DECRYPTED MESSAGE IS\n");
for(i=0;m[i]!=-1;i++)
printf("%c",m[i]);
}
```

OUTPUT:



```
ENTER FIRST PRIME NUMBER
7

ENTER ANOTHER PRIME NUMBER
13

ENTER MESSAGE
hello

POSSIBLE VALUES OF e AND d ARE

5       29
11      59
17      17
19      19
23      47
29      5
31      7
THE ENCRYPTED MESSAGE IS
h⌂ååº
THE DECRYPTED MESSAGE IS
hello
```

EX. NO: 2(C)

IMPLEMENTATION OF DIFFIE HELLMAN KEY EXCHANGE ALGORITHM

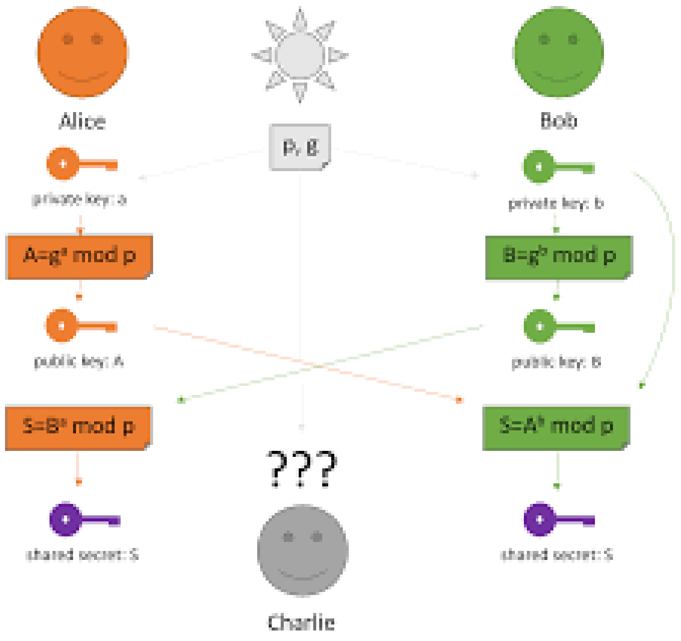
AIM:

To implement the Diffie-Hellman Key Exchange algorithm using C language.

DESCRIPTION:

Diffie–Hellman Key Exchange establishes a shared secret between two parties that can be used for secret communication for exchanging data over a public network. It is primarily used as a method of exchanging cryptography keys for use in symmetric encryption algorithms like AES. The algorithm in itself is very simple. The process begins by having the two parties, Alice and Bob. Let's assume that Alice wants to establish a shared secret with Bob.

EXAMPLE:



ALGORITHM:

STEP-1: Both Alice and Bob shares the same public keys g and p.

STEP-2: Alice selects a random public key a.

STEP-3: Alice computes his secret key A as g^a mod p.

STEP-4: Then Alice sends A to Bob.

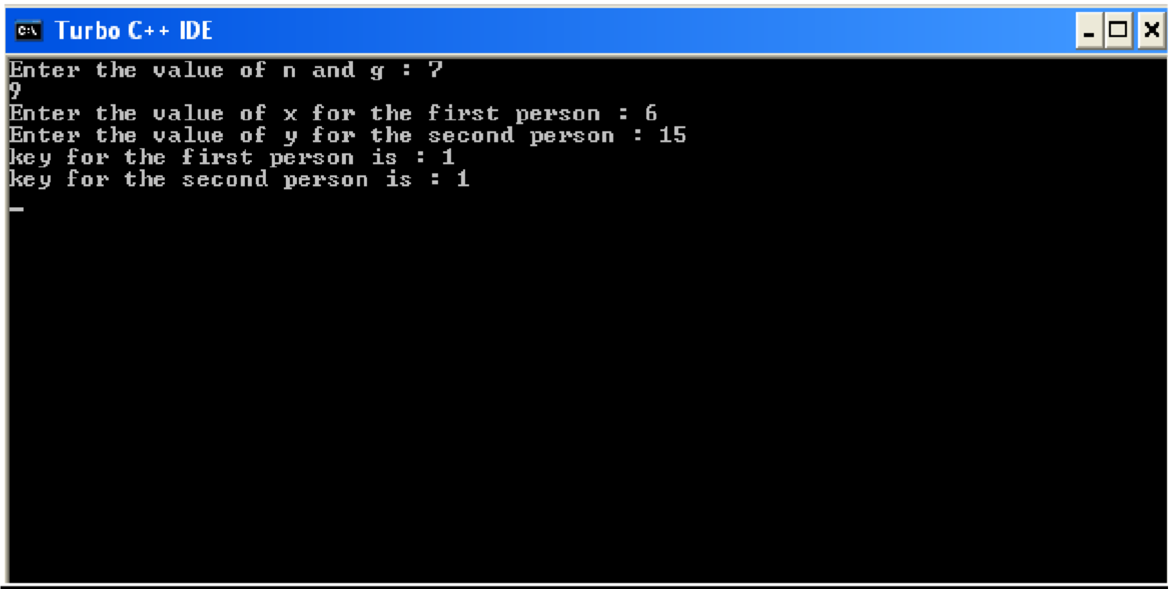
STEP-5: Similarly Bob also selects a public key b and computes his secret key as B and sends the same back to Alice.

STEP-6: Now both of them compute their common secret key as the other one's secret key power of a mod p.

PROGRAM: (Diffie Hellman Key Exchange)

```
#include<stdio.h>
#include<conio.h>
long long int power(int a, int b, int mod)
{
     long long int t;
     if(b==1)
     return a;
     t=power(a,b/2,mod);
     if(b%2==0)
     return (t*t)%mod;
     else
     return (((t*t)%mod)*a)%mod;
}
long int calculateKey(int a, int x, int n)
{
     return power(a,x,n);
}
void main()
{
     int n,g,x,a,y,b;
     clrscr();
     printf("Enter the value of n and g : ");
     scanf("%d%d",&n,&g);
     printf("Enter the value of x for the first person : ");
     scanf("%d",&x);
     a=power(g,x,n);
     printf("Enter the value of y for the second person : ");
     scanf("%d",&y);
     b=power(g,y,n);
     printf("key for the first person is :
     %lld\n",power(b,x,n));
     printf("key for the second person is :
     %lld\n",power(a,y,n));
     getch();
}
```

OUTPUT:



```
Enter the value of n and g : 7
9
Enter the value of x for the first person : 6
Enter the value of y for the second person : 15
key for the first person is : 1
key for the second person is : 1
_
```

EX. NO: 2(D)

IMPLEMENTATION OF MD5

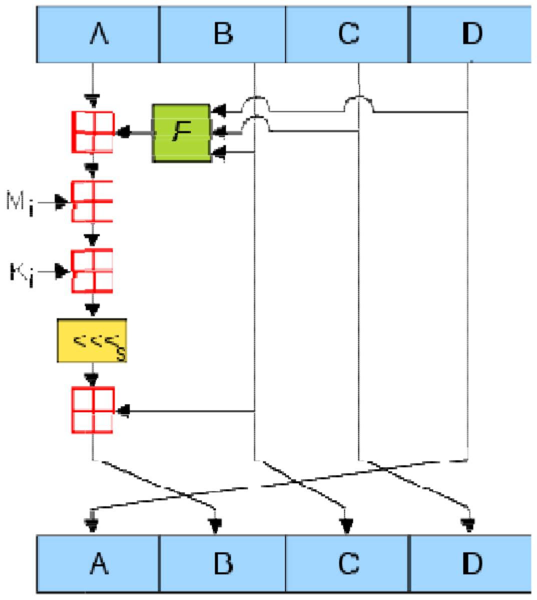
AIM:

To write a C program to implement the MD5 hashing technique.

DESCRIPTION:

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks. The message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo 2^{64}.The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, and D. These are initialized to certain fixed constants. The main algorithm then uses each 512-bit message block in turn to modify the state

EXAMPLE:



ALGORITHM:

STEP-1: Read the 128-bit plain text.

STEP-2: Divide into four blocks of 32-bits named as A, B, C and D.

STEP-3: Compute the functions f, g, h and i with operations such as, rotations, permutations, etc,.

STEP-4: The output of these functions are combined together as F and performed circular shifting and then given to key round.

STEP-5: Finally, right shift of ‘s’ times are performed and the results are combined together to produce the final output.

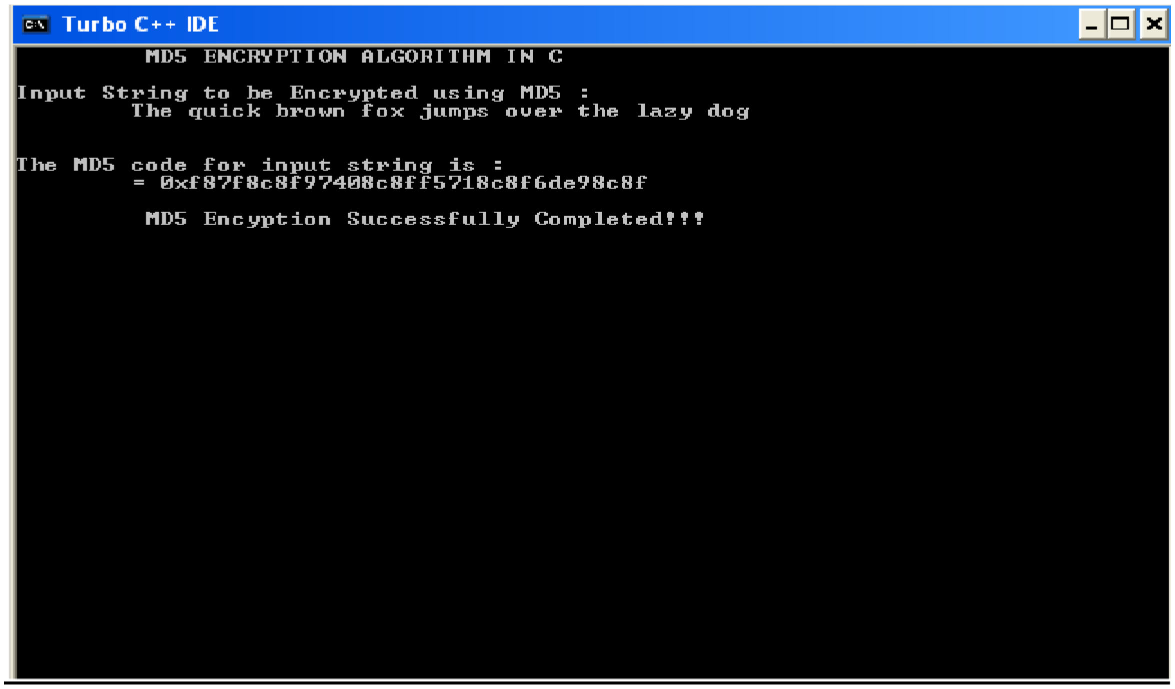
<u>PROGRAM:</u>(MD5)

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include<conio.h>
typedef union uwb
{
    unsigned w;
    unsigned char  b[4];
} MD5union;
     typedef unsigned DigestArray[4];
     unsigned func0( unsigned abcd[]  ){
     return ( abcd[1] & abcd[2]) | (~abcd[1] & abcd[3]);}
     unsigned func1( unsigned abcd[] ){
     return ( abcd[3] & abcd[1]) | (~abcd[3] & abcd[2]);}
     unsigned func2( unsigned abcd[] ){
     return  abcd[1] ^ abcd[2] ^ abcd[3];}
     unsigned func3( unsigned abcd[] ){
     return abcd[2] ^ (abcd[1] |~  abcd[3]);}
     typedef unsigned (*DgstFctn)(unsigned a[]);
unsigned *calctable( unsigned *k)
{
     double s, pwr;
     int i;
     pwr = pow( 2, 32);
for (i=0; i<64; i++)
{
     s = fabs(sin(1+i));
     k[i] = (unsigned)( s * pwr );
}
     return k;
}
unsigned rol( unsigned r, short N )
{
    unsigned  mask1 = (1<<N) -1;
    return ((r>>(32-N)) & mask1) | ((r<<N) &  ~mask1);
}
```

```
unsigned *md5( const char *msg, int mlen)
{
    static DigestArray h0 = { 0x67452301, 0xEFCDAB89,
    0x98BADCFE, 0x10325476 };
    static DgstFctn ff[] = { &func0, &func1, &func2, &func3};
    static short M[] = { 1, 5, 3, 7  };
    static short O[] = { 0, 1, 5, 0 };
    static short rot0[] = { 7,12,17,22};
    static short rot1[] = { 5, 9,14,20};
    static short rot2[] = { 4,11,16,23};
    static short rot3[] = {  6,10,15,21};
    static short *rots[] = {rot0, rot1, rot2, rot3 };
    static unsigned kspace[64];
    static unsigned *k;
    static DigestArray h;
    DigestArray abcd;
    DgstFctn fctn;
    short m, o, g;
    unsigned f;
    short *rotn;
    union
    {
         unsigned w[16];
         char     b[64];
    }mm;
         int os = 0;
         int grp, grps, q, p;
         unsigned char  *msg2;
         if (k==NULL) k= calctable(kspace);
    for (q=0; q<4; q++) h[q]  = h0[q];   // initialize
    {
          grps  = 1 + (mlen+8)/64;
          msg2 = malloc( 64*grps);
          memcpy( msg2, msg,  mlen);
          msg2[mlen] = (unsigned char)0x80;
          q = mlen + 1;
     while (q < 64*grps){ msg2[q] = 0; q++ ;  }
     {
          MD5union u;
          u.w = 8*mlen;
          q -= 8;
          memcpy(msg2+q, &u.w, 4 );
     }
     }
     for (grp=0; grp<grps; grp++)
     {
          memcpy( mm.b, msg2+os, 64);
```

```
        for(q=0;q<4;q++) abcd[q] = h[q];
        for (p = 0; p<4; p++)
        {
             fctn = ff[p];
             rotn = rots[p];
             m = M[p]; o= O[p];
        for (q=0; q<16; q++)
        {
            g = (m*q + o) % 16;
            f = abcd[1] + rol( abcd[0]+ fctn(abcd)+k[q+16*p]
            + mm.w[g], rotn[q%4]);
            abcd[0] = abcd[3];
            abcd[3] = abcd[2];
            abcd[2] = abcd[1];
           abcd[1] = f;
        }}
 for (p=0; p<4; p++)
 h[p] += abcd[p];
 os += 64;
 }
return h;}
void main()
{
     int j,k;
     const char *msg = "The quick brown fox jumps over
     the lazy dog";
     unsigned *d = md5(msg, strlen(msg));
     MD5union u;
     clrscr();
     printf("\t MD5 ENCRYPTION ALGORITHM IN C  \n\n");
     printf("Input String to be Encrypted using MD5  :
     \n\t%s",msg);
     printf("\n\nThe MD5 code for input string is: \n");
     printf("\t= 0x");
     for (j=0;j<4; j++){
     u.w = d[j];
     for (k=0;k<4;k++) printf("%02x",u.b[k]);
}
printf("\n");
printf("\n\t MD5 Encyption Successfully
Completed!!!\n\n");
getch();
system("pause");
getch();}
```

OUTPUT:



```
          MD5 ENCRYPTION ALGORITHM IN C

Input String to be Encrypted using MD5 :
         The quick brown fox jumps over the lazy dog


The MD5 code for input string is :
         = 0xf87f8c8f97408c8ff5718c8f6de98c8f

          MD5 Encyption Successfully Completed!!!
```

EX. NO: 2(E)

IMPLEMENTATION OF SHA-I

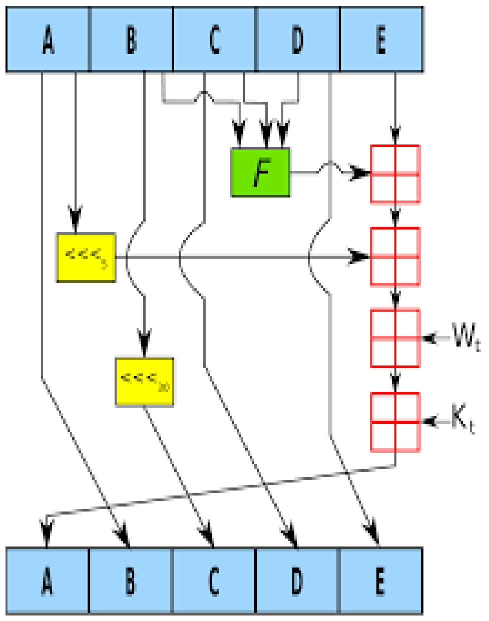
AIM:

To implement the SHA-I hashing technique using C program.

DESCRIPTION:

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function. SHA-1 produces a 160-bit hash value known as a message digest. The way this algorithm works is that for a message of size < 264 bits it computes a 160-bit condensed output called a message digest. The SHA-1 algorithm is designed so that it is practically infeasible to find two input messages that hash to the same output message. A hash function such as SHA-1 is used to calculate an alphanumeric string that serves as the cryptographic representation of a file or a piece of data. This is called a digest and can serve as a digital signature. It is supposed to be unique and non-reversible.

EXAMPLE:



ALGORITHM:

STEP-1: Read the 256-bit key values.

STEP-2: Divide into five equal-sized blocks named A, B, C, D and E.

STEP-3: The blocks B, C and D are passed to the function F.

STEP-4: The resultant value is permuted with block E.

STEP-5: The block A is shifted right by ‘s’ times and permuted with the result of step-4.

STEP-6: Then it is permuted with a weight value and then with some other key pair and taken as the first block.

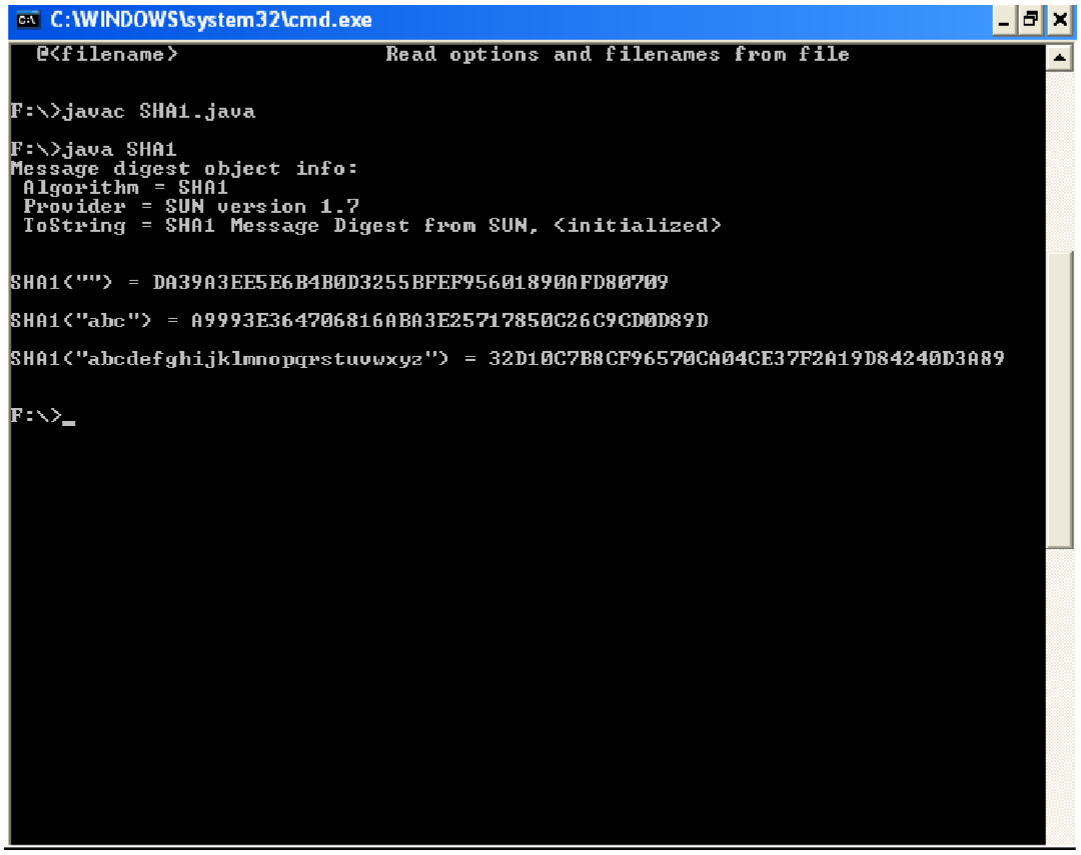
STEP-7: Block A is taken as the second block and the block B is shifted by ‘s’ times and taken as the third block.

STEP-8: The blocks C and D are taken as the block D and E for the final output.

PROGRAM: (Secure Hash Algorithm)

```
import java.security.*;
public class SHA1 {
     public static void main(String[] a) {
     try {
     MessageDigest md = MessageDigest.getInstance("SHA1");
     System.out.println("Message digest object info: ");
     System.out.println(" Algorithm = " +md.getAlgorithm());
     System.out.println(" Provider = " +md.getProvider());
     System.out.println(" ToString = " +md.toString());
     String input = "";
     md.update(input.getBytes());
     byte[] output = md.digest();
     System.out.println();
     System.out.println("SHA1(\""+input+"\") =
     +bytesToHex(output));
     input = "abc";
     md.update(input.getBytes());
     output = md.digest();
     System.out.println();
     System.out.println("SHA1(\""+input+"\") = "
     +bytesToHex(output));
     input = "abcdefghijklmnopqrstuvwxyz";
     md.update(input.getBytes());
     output = md.digest();
     System.out.println();
     System.out.println("SHA1(\"" +input+"\") = "
     +bytesToHex(output));
     System.out.println(""); }
     catch (Exception e) {
     System.out.println("Exception: " +e);
}
}
public static String bytesToHex(byte[] b)
{
     char hexDigit[] = {'0', '1', '2', '3', '4', '5',  '6',
     '7', '8', '9', 'A', 'B', 'C', 'D', 'E',  'F'};
     StringBuffer buf = new StringBuffer();
     for (int j=0; j<b.length; j++) {
     buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
     buf.append(hexDigit[b[j] & 0x0f]); }
     return buf.toString(); }
}
```

OUTPUT:



C:\WINDOWS\system32\cmd.exe

```
  @<filename>                 Read options and filenames from file

F:\>javac SHA1.java

F:\>java SHA1
Message digest object info:
 Algorithm = SHA1
 Provider = SUN version 1.7
 ToString = SHA1 Message Digest from SUN, <initialized>


SHA1("") = DA39A3EE5E6B4B0D3255BFEF95601890AFD80709

SHA1("abc") = A9993E364706816ABA3E25717850C26C9CD0D89D

SHA1("abcdefghijklmnopqrstuvwxyz") = 32D10C7B8CF96570CA04CE37F2A19D84240D3A89


F:\>_
```

EX. NO: 3

IMPLEMENTATION OF DIGITAL SIGNATURE STANDARD

AIM:

To write a C program to implement the signature scheme named digital signature standard (Euclidean Algorithm).

ALGORITHM:

STEP-1: Alice and Bob are investigating a forgery case of x and y.

STEP-2: X had document signed by him but he says he did not sign that document digitally.

STEP-3: Alice reads the two prime numbers p and a.

STEP-4: He chooses a random co-primes alpha and beta and the x's original signature x.

STEP-5: With these values, he applies it to the elliptic curve cryptographic equation to obtain y.

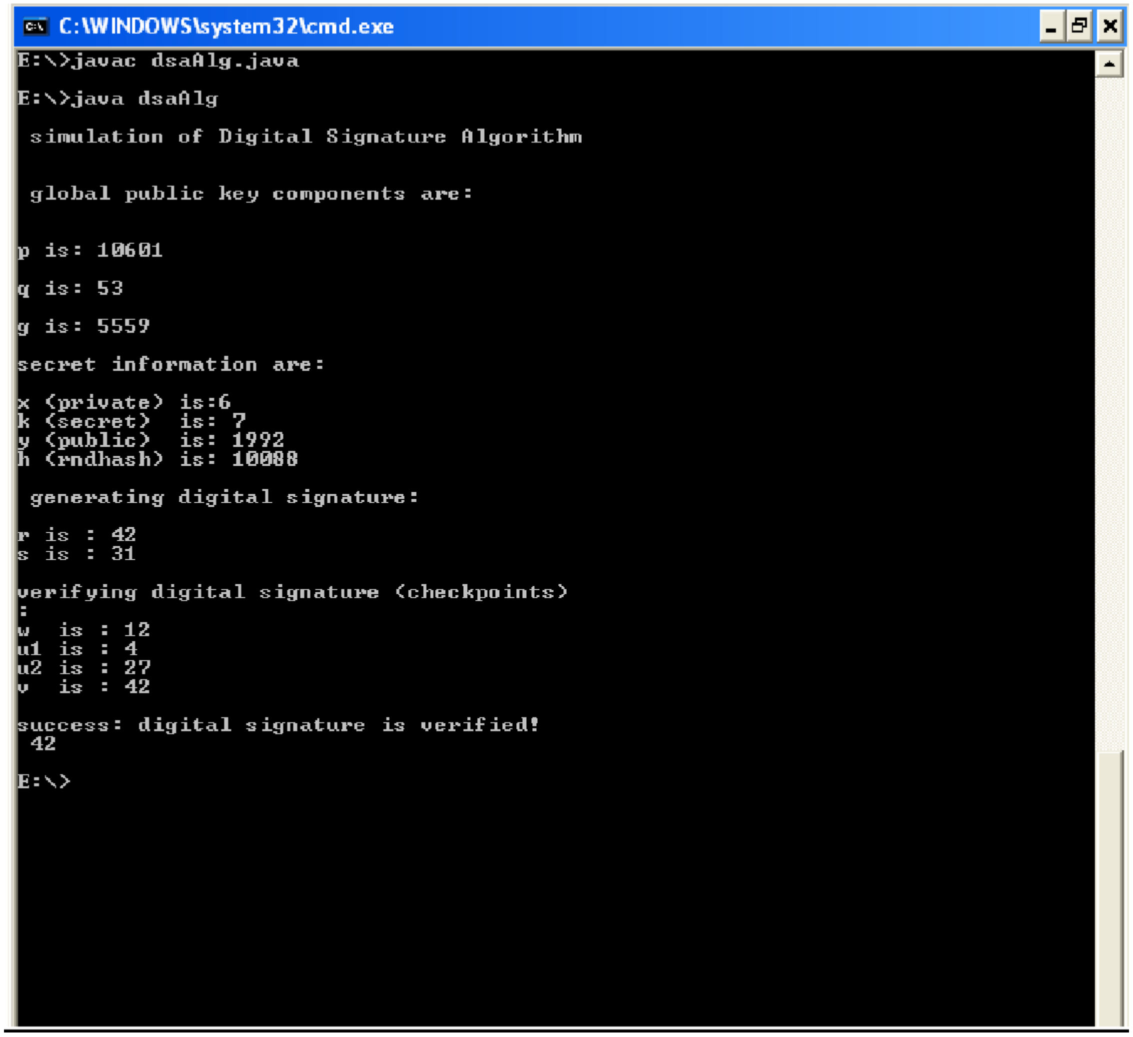
STEP-6: Comparing this 'y' with actual y's document, Alice concludes that y is a forgery.

PROGRAM: (Digital Signature Standard)

```
import java.util.*;
import java.math.BigInteger;
class dsaAlg {
     final static BigInteger one  = new BigInteger("1");
     final static BigInteger zero = new BigInteger("0");
public static BigInteger getNextPrime(String ans)
{
     BigInteger test = new BigInteger(ans);
while (!test.isProbablePrime(99))
e:
{
     test = test.add(one);
}
     return test;
}
public static BigInteger findQ(BigInteger n)
{
     BigInteger start = new BigInteger("2");
while (!n.isProbablePrime(99))
{
     while (!((n.mod(start)).equals(zero)))
     {
          start = start.add(one);
     }
     n = n.divide(start);
}
     return n;
}
public static BigInteger getGen(BigInteger p, BigInteger q,
Random r)
{
     BigInteger h = new BigInteger(p.bitLength(), r);
     h = h.mod(p);
     return h.modPow((p.subtract(one)).divide(q), p);
}
public static void main (String[] args) throws
java.lang.Exception
{
     Random randObj = new Random();
     BigInteger p = getNextPrime("10600");  /* approximate
     prime */
     BigInteger q = findQ(p.subtract(one));
     BigInteger g = getGen(p,q,randObj);
     System.out.println(" \n simulation of Digital Signature
     Algorithm \n");
     System.out.println(" \n global public key components
     are:\n");
     System.out.println("\np is: " + p);
     System.out.println("\nq is: " + q);
     System.out.println("\ng is: " + g);
     BigInteger x = new BigInteger(q.bitLength(), randObj);
     x = x.mod(q);
     BigInteger y = g.modPow(x,p);
     BigInteger k = new BigInteger(q.bitLength(), randObj);
     k = k.mod(q);
     BigInteger r = (g.modPow(k,p)).mod(q);
     BigInteger hashVal = new BigInteger(p.bitLength(),
     randObj);
     BigInteger kInv = k.modInverse(q);
     BigInteger s = kInv.multiply(hashVal.add(x.multiply(r)));
     s = s.mod(q);
     System.out.println("\nsecret information are:\n");
     System.out.println("x (private) is:" + x);
     System.out.println("k (secret)  is: " + k);
     System.out.println("y (public)  is: " + y);
     System.out.println("h (rndhash) is: " + hashVal);
     System.out.println("\n generating digital signature:\n");
     System.out.println("r is : " + r);
     System.out.println("s is : " + s);
     BigInteger w = s.modInverse(q);
     BigInteger u1 = (hashVal.multiply(w)).mod(q);
     BigInteger u2 = (r.multiply(w)).mod(q);
     BigInteger v = (g.modPow(u1,p)).multiply(y.modPow(u2,p));
     v = (v.mod(p)).mod(q);
     System.out.println("\nverifying digital signature
     (checkpoints)\n:");
     System.out.println("w  is : " + w);
```

```
        System.out.println("u1 is : " + u1);
        System.out.println("u2 is : " + u2);
        System.out.println("v is : " + v);
if (v.equals(r))
{
        System.out.println("\nsuccess: digital signature is
        verified!\n " + r);
}
else
{
        System.out.println("\n error: incorrect digital
        signature\n ");
}
}
}
```

OUTPUT:



```
E:\>javac dsaAlg.java

E:\>java dsaAlg

 simulation of Digital Signature Algorithm


 global public key components are:


p is: 10601

q is: 53

g is: 5559

secret information are:

x (private) is:6
k (secret)  is: 7
y (public)  is: 1992
h (rndhash) is: 10088

 generating digital signature:

r is : 42
s is : 31

verifying digital signature (checkpoints)
:
w  is : 12
u1 is : 4
u2 is : 27
v  is : 42

success: digital signature is verified!
 42

E:\>
```