Information Security

PRACTICAL FILE

Submitted by,

Divya A.
College Roll No: 20201457
Exam Roll No: 20016570005
BSc (Hons) Computer Science
Ramanujan College

Submitted to,

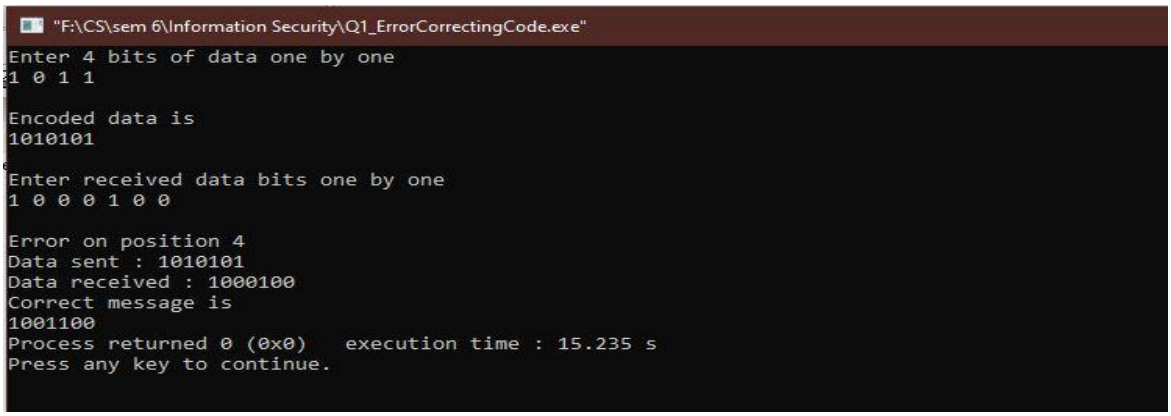
Mr. Sahil Pathak
Assistant Professor
Dept. of Computer Science
Ramanujan College
University of Delhi

TABLE OF CONTENTS

1. Implement the error correcting code.

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

int main() {
   int data[10];
   int dataatrec[10],c,c1,c2,c3,i;

   cout<<"Enter 4 bits of data one by one\n";
   cin>>data[0];
   cin>>data[1];
   cin>>data[2];
   cin>>data[4];

   //Calculation of even parity
   data[6]=data[0]^data[2]^data[4];
data[5]=data[0]^data[1]^data[4];
data[3]=data[0]^data[1]^data[2];

cout<<"\nEncoded data is\n";
for(i=0;i<7;i++)
      cout<<data[i];

cout<<"\n\nEnter received data bits one by one\n";
   for(i=0;i<7;i++)
      cin>>dataatrec[i];

   c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];
c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];
c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];
c=c3*4+c2*2+c1 ;
```

```
   if(c==0) {
cout<<"\nNo error while transmission of data\n";
   }
else {
cout<<"\nError on position "<<c;
cout<<"\nData sent : ";
for(i=0;i<7;i++)
        cout<<data[i];

cout<<"\nData received : ";
       for(i=0;i<7;i++)
        cout<<dataatrec[i];

cout<<"\nCorrect message is\n";
       if(dataatrec[7-c]==0)
dataatrec[7-c]=1;
       else
dataatrec[7-c]=0;
for (i=0;i<7;i++) {
cout<<dataatrec[i];
}
}
return 0;
}
```



2. Implement the error detecting code.

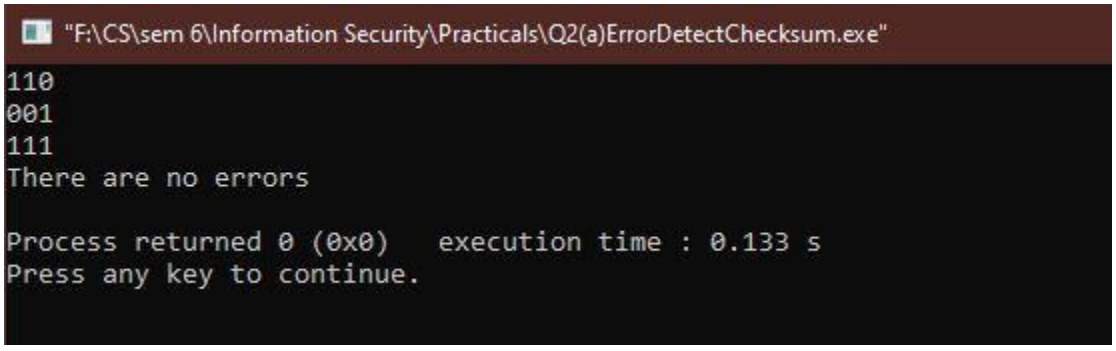
Using Checksum

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

string addBinary(string a, string b)
{
   string result = "";
   int s = 0;

   int i = a.size() - 1, j = b.size() - 1;
   while (i >= 0 || j >= 0 || s == 1)
   {
      s += ((i >= 0)? a[i] - '0': 0);
      s += ((j >= 0)? b[j] - '0': 0);

      result = char(s % 2 + '0') + result;

      s /= 2;

      i--; j--;
   }

   return result;
}

string checkFront(string str)
{
   string start_str = "x";
```

```
    while(start_str != "")
    {
        int size = str.size();
        int size_needed = size - 3;
        if(size_needed>0)
        {
            for(int a = 0; a < size_needed; a++)
            {
                start_str += *str.begin();
                str.erase (str.begin());
            }
            str = addBinary(start_str, str);
        }
        else
         start_str = "";
    }
    return str;

}



string threeDivider(string given_string, bool isSecond)
{
    string str = given_string;
    string new_string = "";
    string old_string = "";
    string sum = "";

    int counter = 0;
```

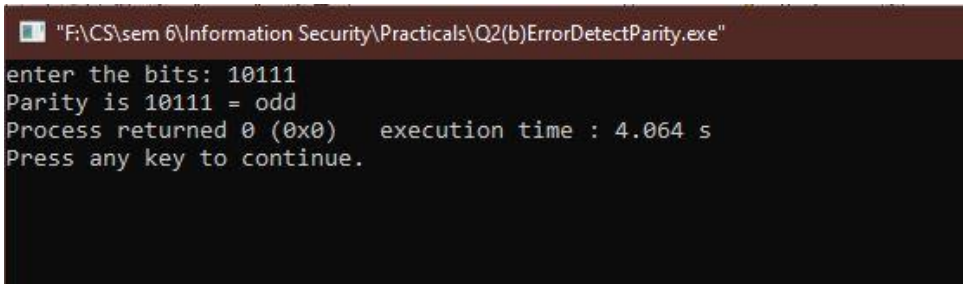
```
   for (string::iterator  j = str.begin(); j != str.end(); ++j)
   {
      if(counter !=3){ new_string += *j; counter++;}

      if(counter ==3)
      {
         sum = addBinary(old_string, new_string);
         old_string = new_string;
         new_string = "";
         counter = 0;
      }
   }
   sum = checkFront(sum);

   if(!isSecond){
      for (string::iterator  j = sum.begin(); j != sum.end(); ++j)
      {
         char new_char = *j;
         if(new_char== '1') new_string +="0";
         else new_string +="1";
      }
      return new_string;
   }
   else
   return sum;

}

int main()
{
```

```
    string str = "100100";
    string complement = threeDivider(str, false);
    cout<<complement<<endl;
    string end_value = threeDivider(str, true);
    cout<<end_value<<endl;
    string temp_result = addBinary(complement, end_value);
    cout << stoi(temp_result) << endl;
    string result = "";
    for (string::iterator  j = temp_result.begin(); j != temp_result.end(); ++j)
       {
          char new_char = *j;
          if(new_char== '1') result +="0";
          else result +="1";



       }
    if(stoi(result) == 0){
       cout << "There are no errors"<< endl;
    }
    else cout << "There is an error."<< endl;
}
```



Using Parity

```
# include<bits/stdc++.h>

# define bool int

using namespace std;

// Function to get parity of number n. It returns 1 if n has odd parity, and returns 0 if n has even
//parity

bool getParity(unsigned int n)

{

   bool parity = 0;

   while (n)

   {

      parity = !parity;

      n     = n & (n - 1);

   }

   return parity;

}

/* Driver program to test getParity() */

int main()

{

   unsigned int n ;

   cout<<"enter the bits: ";

   cin>>n;

   cout<<"Parity is "<<n<<" = "<<(getParity(n)? "odd": "even");

   getchar();

   return 0;

}
```

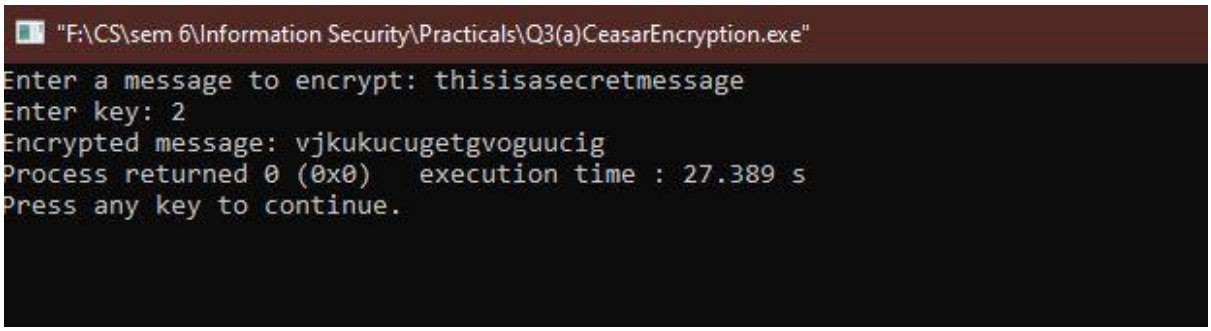


3. Implement caeser cipher substitution operation.

Encryption

```
#include<iostream>


using namespace std;


int main()
{
char message[100], ch;
int i, key;
cout << "Enter a message to encrypt: ";
cin.getline(message, 100);
cout << "Enter key: ";
cin >> key;
for(i = 0; message[i] != '\0'; ++i){
ch = message[i];
if(ch >= 'a' && ch <= 'z'){
ch = ch + key;
if(ch > 'z'){
ch = ch - 'z' + 'a' - 1;
}
message[i] = ch;
}
else if(ch >= 'A' && ch <= 'Z'){
ch = ch + key;
if(ch > 'Z'){
ch = ch - 'Z' + 'A' - 1;
}
message[i] = ch;
}
}
```

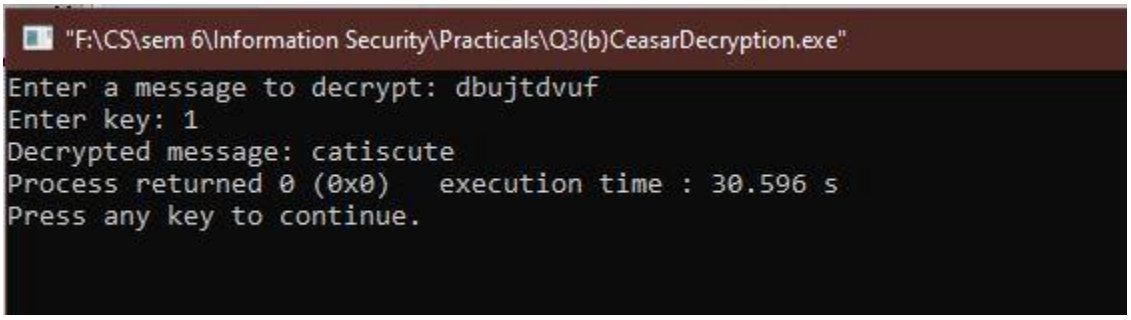
```
cout << "Encrypted message: " << message;

return 0;

}
```



Decryption

```
#include<iostream>


using namespace std;

int main()

{

char message[100], ch;

int i, key;

cout << "Enter a message to decrypt: ";

cin.getline(message, 100);

cout << "Enter key: ";

cin >> key;

for(i = 0; message[i] != '\0'; ++i){

ch = message[i];

if(ch >= 'a' && ch <= 'z'){

ch = ch - key;

if(ch < 'a'){

ch = ch + 'z' - 'a' + 1;

}

message[i] = ch;

}
```

```
else if(ch >= 'A' && ch <= 'Z'){
ch = ch - key;
if(ch > 'a'){
ch = ch + 'Z' - 'A' + 1;
}
message[i] = ch;
}
}
cout << "Decrypted message: " << message;
return 0;
}
```



4. Implement monoalphabetic and polyalphabetic cipher substitution operation.

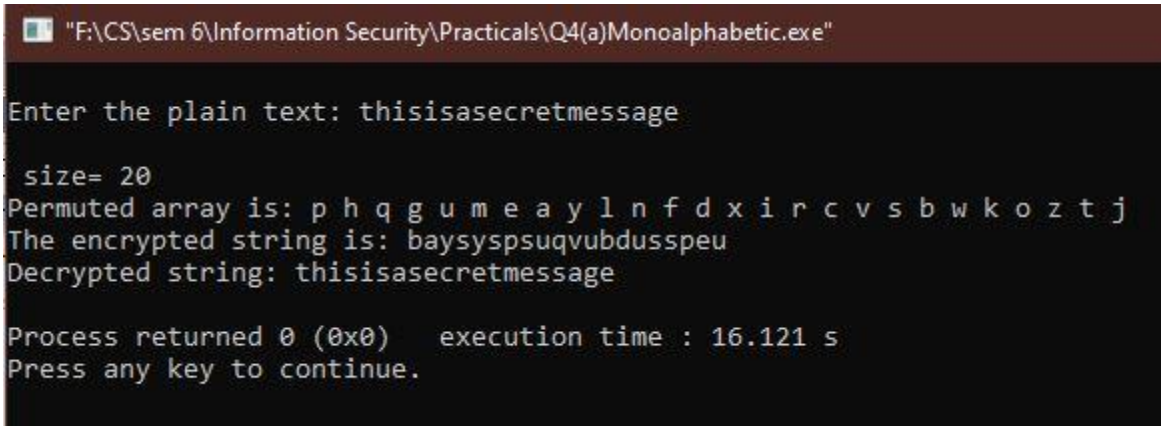
Monoalphabetic Cipher

```
#include<stdio.h>

#include<conio.h>
#include<stdlib.h>
#include<string.h>

main()
{

char plain[200],encr[200],encr1[200],decr[200];
```

```
int ch,flag[26];
char ch1;
char per[26];
int i,j,size,k=0,a=0;
char pos;

printf("\nEnter the plain text: ");
gets(plain);
size=strlen(plain);
printf("\n size= %d",size);
 for(i=0;i<26;i++)
  flag[i]=0;

 for(i=0;i<26;i++)
 {
   a=rand()%26;
  while(flag[a]==1)
   a=rand()%26;
  flag[a]=1;
  per[i]=(char)(a+97);
 }
printf("\nPermuted array is: ");
for(i=0;i<26;i++)
{
printf("%c ",per[i]);
}
for(j=0;j<size;j++)
{
ch=plain[j];
ch=ch-97;
ch1=per[ch];
```

```
encr1[j]=ch1;
}

//encr1[25]='\0';
//print

printf("\nThe encrypted string is: ");
for(i=0;i<size;i++)
{
printf("%c",encr1[i]);
}



//decryption
 i=0;
 while(i<size)
 {
  //a=ct[i];
  //a=a-97;
  //pt[i]=key[a];
  a=0;
  while(a!=26)
  {
   if(per[a]==encr1[i])
   {
    decr[i]=a+97;
    break;
   }
   a++;
  }
```

```
 i++;
 }
 decr[i]='\0';
 printf("\nDecrypted string: %s\n",decr);
 getch();
}
```

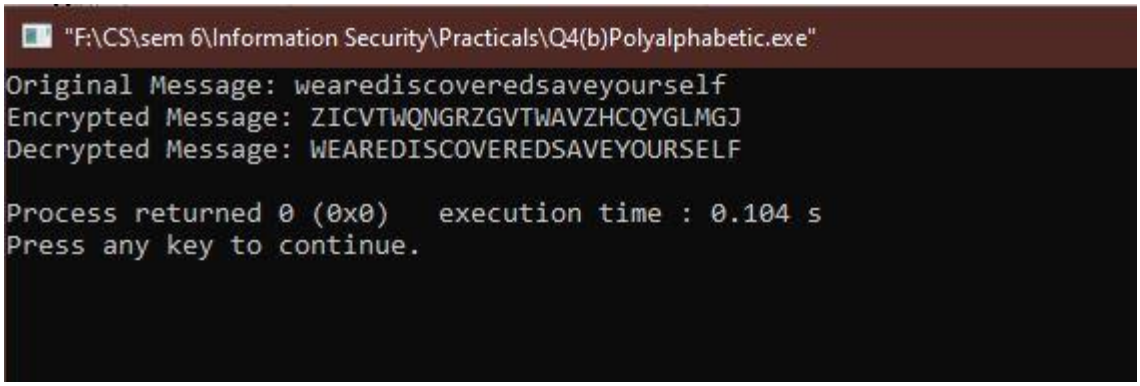


Polyalphabetic Cipher

```
#include <iostream>
#include <string>
using namespace std;
class Vig {
  public:
    string k;
  Vig(string k) {
    for (int i = 0; i < k.size(); ++i) {
      if (k[i] >= 'A' && k[i] <= 'Z')
        this->k += k[i];
      else if (k[i] >= 'a' && k[i] <= 'z')
        this->k += k[i] + 'A' - 'a';
    }
  }
  string encryption(string t) {
    string output;
```

```
    for (int i = 0, j = 0; i < t.length(); ++i) {
      char c = t[i];
      if (c >= 'a' && c <= 'z')
        c += 'A' - 'a';
      else if (c < 'A' || c > 'Z')
        continue;
      output += (c + k[j] - 2 * 'A') % 26 + 'A'; //added 'A' to bring it in range of ASCII alphabet [ 65-90 | A-Z ]
      j = (j + 1) % k.length();
    }
    return output;
  }
  string decryption(string t) {
    string output;
    for (int i = 0, j = 0; i < t.length(); ++i) {
      char c = t[i];
      if (c >= 'a' && c <= 'z')
        c += 'A' - 'a';
      else if (c < 'A' || c > 'Z')
        continue;
      output += (c - k[j] + 26) % 26 + 'A';//added 'A' to bring it in range of ASCII alphabet [ 65-90 | A-Z ]
      j = (j + 1) % k.length();
    }
    return output;
  }
};
int main() {
  Vig v("deceptive");
  string ori ="wearediscoveredsaveyourself";
  string encrypt = v.encryption(ori);
  string decrypt = v.decryption(encrypt);
```

```
  cout << "Original Message: "<<ori<< endl;
  cout << "Encrypted Message: " << encrypt << endl;
  cout << "Decrypted Message: " << decrypt << endl;
}
```



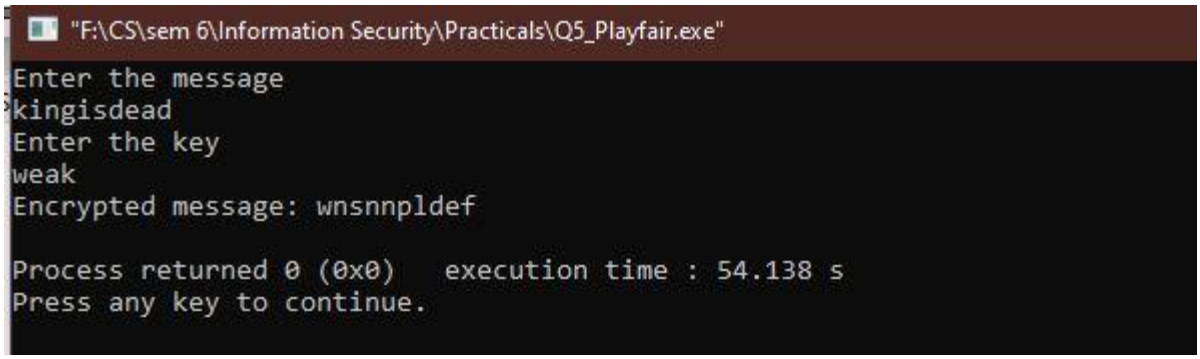
5. Implement playfair cipher substitution operation

```
#include<iostream>
#include<string>
#include<vector>
#include<map>
using namespace std;
int main(){
   int i,j,k,n;
   cout<<"Enter the message"<<endl;
   string s,origin;
   getline(cin,origin);
   cout<<"Enter the key"<<endl;
   string key;
   cin>>key;
   for(i=0;i<origin.size();i++){
      if(origin[i]!=' ')
         s+= origin[i];
   }
   vector<vector<char> > a(5,vector<char>(5,' '));
```

```
n=5;
map<char,int> mp;
k=0;
int pi,pj;
for(i=0;i<n;i++){
   for(j=0;j<n;j++){
      while(mp[key[k]]>0&&k<key.size()){
         k++;
      }
      if(k<key.size()){
         a[i][j]=key[k];
         mp[key[k]]++;
         pi=i;
         pj=j;
      }
      if(k==key.size())
      break;
   }
   if(k==key.size())
      break;
}
k=0;
for(;i<n;i++){
   for(;j<n;j++){
      while(mp[char(k+'a')]>0&&k<26){
         k++;
      }
      if(char(k+'a')=='j'){
         j--;
         k++;
         continue;
```

```
        }
        if(k<26){
            a[i][j]=char(k+'a');
            mp[char(k+'a')]++;
        }
    }
    j=0;
}
string ans;
if(s.size()%2==1)
    s+="x";
for(i=0;i<s.size()-1;i++){
    if(s[i]==s[i+1])
        s[i+1]='x';
}
map<char,pair<int,int> > mp2;
for(i=0;i<n;i++){
    for(j=0;j<n;j++){
        mp2[a[i][j]] = make_pair(i,j);
    }
}

for(i=0;i<s.size()-1;i+=2){
    int y1 = mp2[s[i]].first;
    int x1 = mp2[s[i]].second;
    int y2 = mp2[s[i+1]].first;
    int x2 = mp2[s[i+1]].second;
    if(y1==y2){
        ans+=a[y1][(x1+1)%5];
        ans+=a[y1][(x2+1)%5];
    }
```

```
    else if(x1==x2){
        ans+=a[(y1+1)%5][x1];
        ans+=a[(y2+1)%5][x2];
    }
    else {
        ans+=a[y1][x2];
        ans+=a[y2][x1];
    }
  }
  cout<<"Encrypted message: "<<ans<<'\n';
  return 0;
}
```

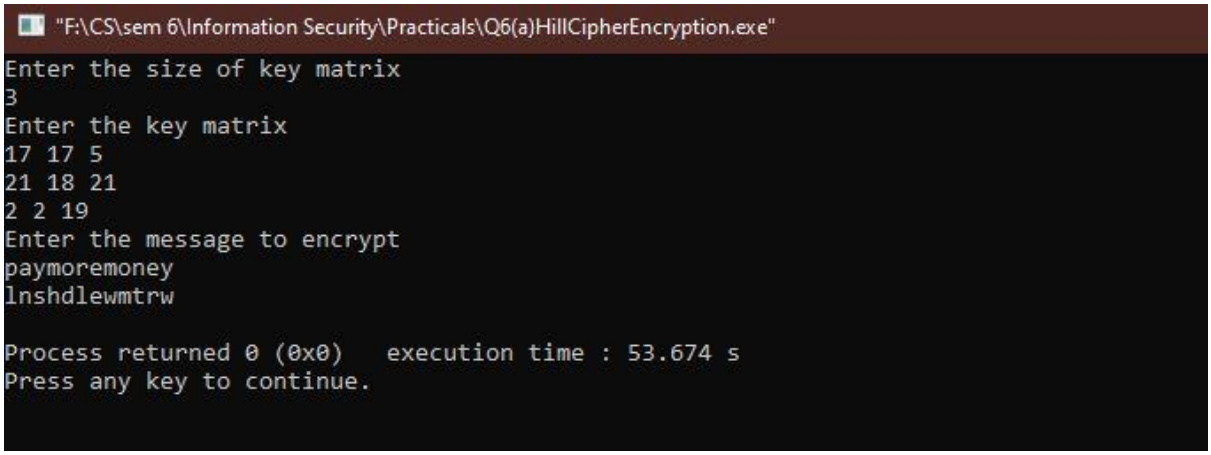


6. Implement hill cipher substitution operation.

Encryption

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
  int x,y,i,j,k,n;
  cout<<"Enter the size of key matrix\n";
  cin>>n;
  cout<<"Enter the key matrix\n";
  int a[n][n];
```

```
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            cin>>a[i][j];
        }
    }
    cout<<"Enter the message to encrypt\n";
    string s;
    cin>>s;
    int temp = (n-s.size()%n)%n;
    for(i=0;i<temp;i++){
        s+='x';
    }
    k=0;
    string ans="";
    while(k<s.size()){
        for(i=0;i<n;i++){
            int sum = 0;
            int temp = k;
            for(j=0;j<n;j++){
                sum += (a[i][j]%26*(s[temp++]-'a')%26)%26;
                sum = sum%26;
            }
            ans+=(sum+'a');
        }
        k+=n;
    }
    cout<<ans<<'\n';

    return 0;
}
```



Decryption

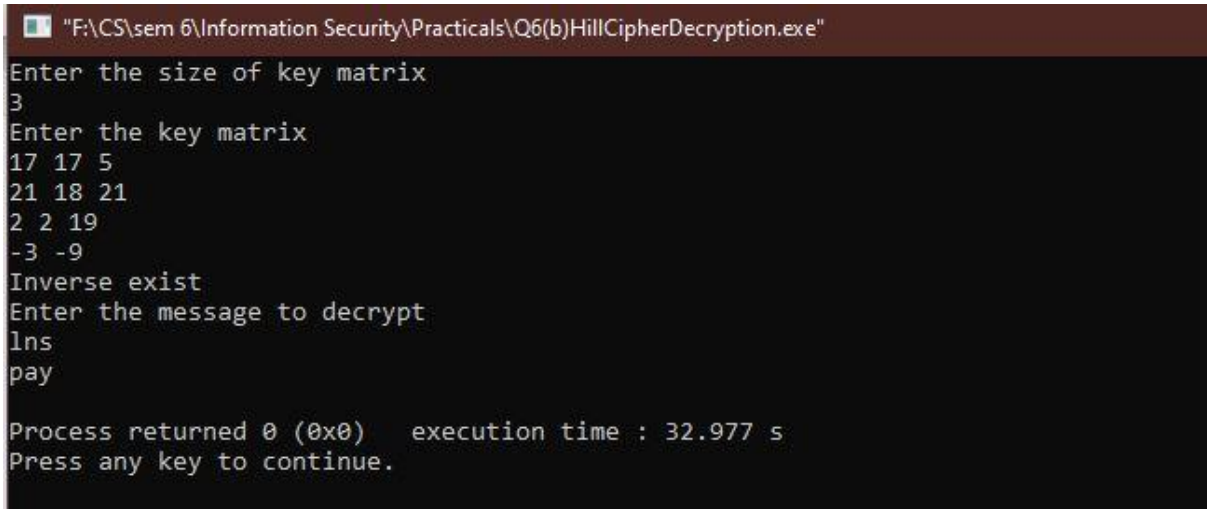
```
#include<iostream>
#include<vector>
using namespace std;

int modInverse(int a, int m){
   a=a%m;
   for(int x=-m;x<m;x++)
     if((a*x)%m==1)
       return x;
}

void getCofactor(vector<vector<int> > &a, vector<vector<int> > &temp, int p, int q, int n){
   int i=0,j=0;
   for(int row=0;row<n;row++){
      for(int col=0;col<n;col++){
         if(row!=p&&col!=q){
            temp[i][j++] = a[row][col];
            if (j==n-1){
               j=0;
               i++;
            }
```

```
        }
      }
   }
}

int determinant(vector<vector<int> > &a, int n, int N){
   int D = 0;
   if(n==1)
      return a[0][0];
   vector<vector<int> > temp(N, vector<int>(N));
   int sign = 1;
   for(int f=0;f<n;f++){
      getCofactor(a, temp, 0, f, n);
      D += sign * a[0][f] * determinant(temp, n - 1, N);
      sign = -sign;
   }
   return D;
}

void adjoint(vector<vector<int> > &a,vector<vector<int> > &adj,int N){
   if(N == 1){
      adj[0][0] = 1;
      return;
   }
   int sign = 1;
   vector<vector<int> > temp(N, vector<int>(N));
   for(int i=0;i<N;i++){
      for(int j=0;j<N;j++){
         getCofactor(a, temp, i, j, N);
         sign = ((i+j)%2==0)? 1: -1;
         adj[j][i] = (sign)*(determinant(temp, N-1 , N));
```

```
        }
    }
}

bool inverse(vector<vector<int> > &a, vector<vector<int> > &inv, int N){
    int det = determinant(a, N, N);
    if(det == 0){
        cout << "Inverse does not exist";
        return false;
    }
    int invDet = modInverse(det,26);
    cout<<det%26<<' '<<invDet<<'\n';
    vector<vector<int> > adj(N, vector<int>(N));
    adjoint(a, adj, N);
    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++)
            inv[i][j] = (adj[i][j]*invDet)%26;
    return true;
}


int main(){
    int x,y,i,j,k,n;
    cout<<"Enter the size of key matrix\n";
    cin>>n;
    cout<<"Enter the key matrix\n";
    vector<vector<int> > a(n, vector<int>(n));
    vector<vector<int> > adj(n, vector<int>(n));
    vector<vector<int> > inv(n, vector<int>(n));

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

```
   for(j=0;j<n;j++){
      cin>>a[i][j];
   }
}
if(inverse(a,inv,n)){
   cout<<"Inverse exist\n";
}

cout<<"Enter the message to decrypt\n";
string s;
cin>>s;
k=0;
string ans;
while(k<s.size()){
   for(i=0;i<n;i++){
      int sum = 0;
      int temp = k;
      for(j=0;j<n;j++){
         sum += ((inv[i][j] + 26)%26*(s[temp++]-'a')%26)%26;
         sum = sum%26;
      }
      ans+=(sum+'a');
   }
   k+=n;
}
//ans+='\0';
int f=ans.size()-1;
while(ans[f]=='x'){
   f--;
}
```

```
    for(i=0;i<=f;i++){
        cout<<ans[i];
    }
    cout<<'\n';
    return 0;
}
```

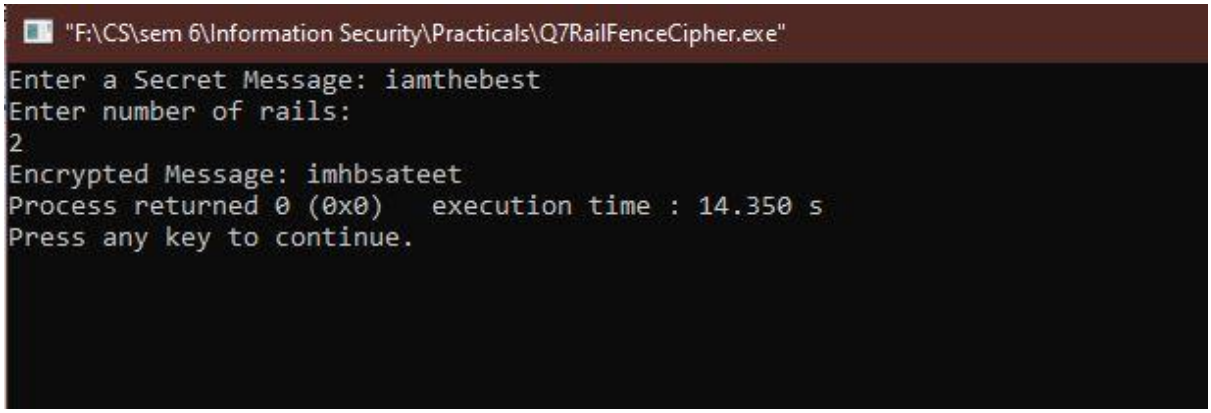


7. Implement rail fence cipher transposition operation.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
main()
{
    int i,j,len,rails,count,code[100][1000];
    char str[1000];
    printf("Enter a Secret Message: ");
    gets(str);
    len=strlen(str);
    printf("Enter number of rails:\n");
    scanf("%d",&rails);
    for(i=0;i<rails;i++)
    {
```

```
      for(j=0;j<len;j++)
      {
         code[i][j]=0;
      }
   }
count=0;
j=0;
while(j<len)
{
if(count%2==0)
{
   for(i=0;i<rails;i++)
   {
      //strcpy(code[i][j],str[j]);
      code[i][j]=(int)str[j];
      j++;
   }

}
else
{

   for(i=rails-2;i>0;i--)
   {
      code[i][j]=(int)str[j];
   j++;
   }
}

count++;
}
```

```
printf("Encrypted Message: ");
for(i=0;i<rails;i++)
{
   for(j=0;j<len;j++)

   {
      if(code[i][j]!=0)

      printf("%c",code[i][j]);
   }

}
}
```



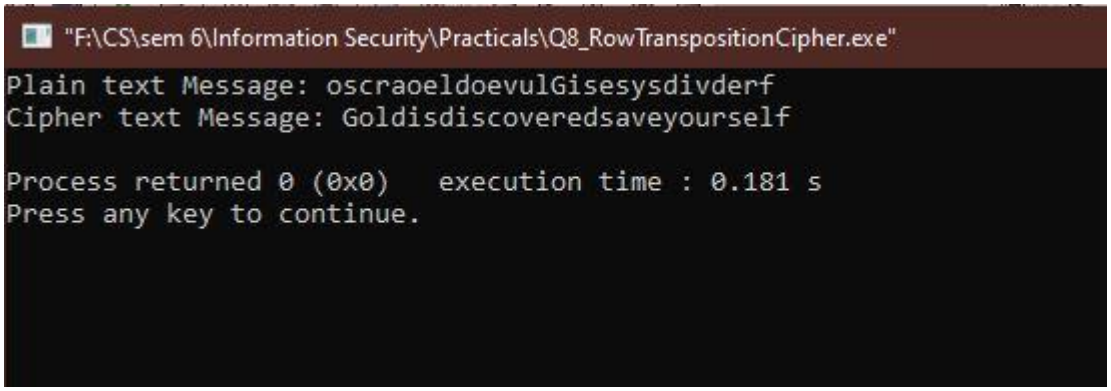
8. Implement row transposition cipher transposition operation.

```
#include<bits/stdc++.h>
using namespace std;
string const key = "HACK";// Key
map<int,int> keyMap;
void setPermutationOrder()
{
        for(int i=0; i < key.length(); i++)
        {
                keyMap[key[i]] = i;
```

```
	}
}
string encryptMessage(string msg)
{
	int row,col,j;
	string cipher = "";
	col = key.length();
	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')
			{
				matrix[i][j] = '_';
				j++;
			}
			if( isalpha(msg[k]) || msg[k]==' ')
			{
				matrix[i][j] = msg[k];
				j++;
			}
			k++;
		}
	}

	for (map<int,int>::iterator ii = keyMap.begin(); ii!=keyMap.end(); ++ii)
```

```
	{
		j=ii->second;
				for (int i=0; i<row; i++)
		{
			if( isalpha(matrix[i][j]) || matrix[i][j]==' ' || matrix[i][j]=='_')
				cipher += matrix[i][j];
		}
	}

	return cipher;
}
string decryptMessage(string cipher)
{
	int col = key.length();
	int row = cipher.length()/col;
	char cipherMat[row][col];
	for (int j=0,k=0; j<col; j++)
		for (int i=0; i<row; i++)
			cipherMat[i][j] = cipher[k++];
	int index = 0;
	for( map<int,int>::iterator ii=keyMap.begin(); ii!=keyMap.end(); ++ii)
		ii->second = index++;
	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];
```

```
            }
        }
        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;
}
int main(void)
{
        string msg = "Goldisdiscoveredsaveyourself";
        setPermutationOrder();
        string cipher = encryptMessage(msg);
        cout << "Plain text Message: " << cipher << endl;
        cout << "Cipher text Message: " << decryptMessage(cipher) << endl;
        return 0;
}
```



9. Implement product cipher transposition operation.

```
#include<stdio.h>
#include<cstring>
#include<cmath>
#include<iostream>
using namespace std;



/* key HEX is : 133457799bbcdff1 */
int key[64]={   0,0,0,1,    0,0,1,1,
                                0,0,1,1, 0,1,0,0,
                                0,1,0,1, 0,1,1,1,
                                0,1,1,1, 1,0,0,1,
                                1,0,0,1, 1,0,1,1,
                                1,0,1,1, 1,1,0,0,
                                1,1,0,1, 1,1,1,1,
                                1,1,1,1, 0,0,0,1
                        };


class Des
{
 public:
  int
keyi[16][48],total[64],left[32],right[32],ck[28],dk[28],expansion[48],z[48],xor1[48],sub[32],p[32],xor
2[32],temp[64],
                pc1[56],ip[64],inv[8][8];


  char final[1000];
  void IP();
  void PermChoice1();
  void PermChoice2();
```

```
 void Expansion();
 void inverseIP();
 void xor_two();
 void xor_oneE(int);
 void xor_oneD(int);
 void substitution();
 void permutation();
 void keygen();
 char * Encrypt(char *);
 char * Decrypt(char *);
};

/*
Initial permutation, algorithmically made.
note that: standard PC-1 goes from 1 to 64
but here to handle index we go from 0 to 63
Data stored stored to ip
*/
void Des::IP()
{
	int k=58,i;
	for(i=0;i<32;i++)
	{
		/* k-1 is done to handle 0th index, kth index of message becomes Ith index of IP
		Traditionally, 58th index of message becomes 1st index of IP, but we need to use index 0 as well,
		so here we use, 57th index of message becomes 0th index of IP*/
		ip[i]=total[k-1];
		if(k-8>0)  k=k-8;
		else	k=k+58;
	}
```

```
	k=57;
	for( i=32;i<64;i++)
	{
		ip[i]=total[k-1];
		if(k-8>0)   k=k-8;
		else        k=k+58;
	}
}

/*
Applied PC-1 to keys
intelligently applied.
stored to pc1
*/
void Des::PermChoice1()
{
	int k=57,i;
	for(i=0;i<28;i++)
	{
		pc1[i]=key[k-1];
		if(k-8>0)    k=k-8;
		else         k=k+57;
	}
	k=63;
	for( i=28;i<52;i++)
	{
		pc1[i]=key[k-1];
		if(k-8>0)    k=k-8;
		else       k=k+55;
	}
```

```
        k=28;
        for(i=52;i<56;i++)
        {
                pc1[i]=key[k-1];
                k=k-8;
        }

}
/*
 * Convert 32-bit message to 48 bit message
 */
void Des::Expansion()
{
   int exp[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
                };
                for(int i=0;i<48;++i){
                  expansion[i]=right[exp[i]-1];
                }
}
/* Applies PC-2 to Key*/
void Des::PermChoice2()
{
        int per[56],i,k;
        // concatenate CkDk to form 'per', then apply PC-2 to it
```

```
        for(i=0;i<28;i++)per[i]=ck[i];
        k=0;
        for(i=28;i<56;i++)          per[i]=dk[k++];

        int PC2[]={ 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
                          };
        // did -1 in PC-2 to start indexes from 0
        for(int i=0;i<48;++i){
                z[i]=per[PC2[i]-1];
        }
}
/*
 * Xor the expanded right half with key (48 bits XOR)
 */
void Des::xor_oneE(int round)
{
        for(int i=0;i<48;i++){
       xor1[i]=expansion[i]^keyi[round-1][i];
     }
}

/*In decryption, subkeys order are opposite
*/
void Des::xor_oneD(int round)
{
 int i;
        for(i=0;i<48;i++)
                xor1[i]=expansion[i]^keyi[16-round][i];
```

```
}

void Des::substitution()
{
        int s[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}}
        };

    int a[8][6],k=0,i,j,p,q,count=0,g=0,v;

/* Dividing entire lenght to 8 rows of 6 bits each*/
    for(i=0;i<8;i++)
    {
            for(j=0;j<6;j++)
            {
                    a[i][j]=xor1[k++];
            }
    }

int EMsgLen=-1;

/* Mapping each row to diff address*/
    for(i=0;i<8;i++)
```

```
        {
                // 0th & 5th gives the Row number of i-th S-Box
                int row=(a[i][0]*2)+(a[i][5]*1);

    // 1st, 2nd, 3rd, 4th bit gives the column number of i-th S-Box
    int col=(a[i][1]*8)+(a[i][2]*4)+(a[i][3]*2)+(a[i][4]*1);

    // extract content- v from the location
    v=s[i][row][col];

    /*
     * This 'v' is in integer format, needs to be converted to 4 binary bits
     * & then replace those 6 bits with these 4 bits
     */
                int d,i=-1,a[4];
                while(v){
                        a[++i]= (v%2);
                        v=v/2;
                }
                while(i<4){
                        a[++i]=0;
                }

    for(i=3;i>=0;i--){
        sub[++EMsgLen]=a[i];
    }
        }
        /* substitution result gets hold into sub[] */
}

/* permutates the result from S-Boxes substitution */
```

```
void Des::permutation()
{
   int perm[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
                  };

   for(int i=0;i<32;i++){
      p[i]=sub[perm[i]-1];
   }
}

/* Performs XOR of Ln-1 and f(R(n-1), Kn)*/
void Des::xor_two()
{
          for(int i=0;i<32;i++){
                    xor2[i]=left[i]^p[i];
          }
}

/*
 * Final permutation, performed at the end of 16 cycles.
 * Inverse of the Initial Permutation IP
 */
void Des::inverseIP()
{
          int p=40,q=8,k1,k2,i,j;
      // for 8 rows
          for(i=0;i<8;i++)
          {
```

```
                k1=p;k2=q;
            // for 8 columns
                for(j=0;j<8;j++){
              // even column
                    if(j%2==0){  inv[i][j]=temp[k1-1];   k1=k1+8; }
                    // odd column
                    else     {  inv[i][j]=temp[k2-1];   k2=k2+8; }
                }
                // for subsequent rows, decrement p,q by 1
                p=p-1;q=q-1;
          }
}

char* Des::Encrypt(char *Text1)
{
 /* creating a copy to perform operations on */
 char *Text=new char[1000];
 char *OmsgHEX=new char[1000];
 int Omsgi=-1;
 char *EmsgHEX=new char[1000];
 int Emsgi=-1;
 strcpy(Text,Text1);

 /* Generate all the required keys
  * The DES is gonna run multiple times to be able to encrypt entire message,
  * so it is logical to generate and store all keys at once.
  */
 keygen();

 int i,a1,j,nB,m,iB,k,K,B[8],n,t,d,round;
```

```
/* Encryption condition: message length should be multiple of eight(64 bits encypted at once),
 * if not multiple of eight, 0 is appended to make it multiple of 8.
 * for eg -
 * let message length is 15. It can be divided into 2 message blocks (8+7), DES needs to be run 2 times.
 * In second block 0 needs to be appended in the HEXA code of plaintext to make it multiple of 8.
 */
int messageBlocks= ceil(strlen(Text)/8.0);

 /*
  * A DES run has 16 cycles. In a complete run it can encrypt 64 bits (8 characters of 8 bit each)
  * The entire message is broken into groups of 8 characters,
  * then DES is RUN on those 8 characters.
  */
   int encryptedMsgIndex=-1;     // acts as index for final   encrypted message
   int msgIndex=-1;                              // acts as index for initial plaintext message
   int binaryIndex;
   for(m=0; m<messageBlocks ;m++){

      /* Converting group of 8-char to binary */
      binaryIndex=-1;
      for(i=0;i<8;i++){
            // convert char to Ascii and then to Binary
            n=(int)Text[++msgIndex];
            K=-1;
            while(n){
                  B[++K]=n%2;
                  n/=2;
            }
            while(K<8){
                  B[++K]=0;
```

```
      }
      /* store in reverse order,
   as, after conversion, binary is always read upside down.
      */
      for(K=7;K>=0;K--){
                total[++binaryIndex]=B[K];
      }
   }
//  cout<<"\nBinary index is : "<<binaryIndex<<" ok \n";
/* if string is not a multiple of 8, append 0 in HEXA */
while(binaryIndex!=63) total[++binaryIndex]=0;

/* Storing Plaintext message as HEX
 */
int sssk=0;
for(i=0;i<binaryIndex;i=i+4){
   sssk=total[i]*8+total[i+1]*4+total[i+2]*2+total[i+3]*1;
                if(sssk<=9){
                        OmsgHEX[++Omsgi]=(char)(sssk+48);
                }else{
                        OmsgHEX[++Omsgi]=(char)(sssk+87);
                }
}
/**/

     /* Initial permutation
 * Runs on the message, total[], to give IP
 */
     IP();
/* copying back updated contents from IP[] to total[] */
for(i=0;i<64;i++) total[i]=ip[i];
```

```
        /* IP is to L0 and R0*/
        for(i=0;i<32;i++) left[i]=total[i];
        for(i=32;i<64;i++) right[i-32]=total[i];

        /* 16 Cycles of DES
      */
        for(round=1;round<=16;round++)
        {
       /* here, the drives are LEFT(n-1) and RIGHT(n-1) */
       Expansion();       /** Convert 32-bit message to 48 bit message using E-bit Selection table  */
       xor_oneE(round);   /** Xor the expanded right half with key (48 bits XOR) */
       substitution();    /** S-Box substitutions take place, result goes to sub[] */
       permutation();     /**  permutates the result from S-Boxes substitution, result goes from sub[]
to p[]*/

       /* The drivers performs 'f' function & resultant is XORed with L(n-1) */
       xor_two();         /** Performs XOR of Ln-1 and f(R(n-1), Kn) */
       /* New Drive R(n) is ready to take the seat, currently resides in xor2[]*/

       /* For the next iteration,
        * New Drive L(n)= old Driver R(n-1)
        */
       for(i=0;i<32;i++) left[i]=right[i];
       /*
        * New Driver R(n) comes and takes seat */
       for(i=0;i<32;i++) right[i]=xor2[i];
      }

     /* concatenate the final result from 16 cycles in the formation - R16,L16 */
        for(i=0;i<32;i++) temp[i]=right[i];
```

```
    for(i=32;i<64;i++)    temp[i]=left[i-32];

/* Applying the final permutation- inverseIP over temp[], result comes in inv[]*/
inverseIP();

/* Converting Binary Matrix to HEX
 * Logic - consider 8 bits as two groups of 4-4 bits, then convert these 4 bits to HEX as
 * (3rd bit *8)+(2nd bit*4)+(1st bit*2)+(0th bit*1)
 * 8 rows i.e. one character per row
 * For each char, 2 groups of 4-4 columns and 1 bit per column.
 */
int sss=0;
for(i=0;i<8;i++){
    for(j=0;j<8;j=j+4){
                sss=inv[i][j]*8+inv[i][j+1]*4+inv[i][j+2]*2+inv[i][j+3]*1;
                if(sss<=9){
               EmsgHEX[++Emsgi]=(char)(sss+48); // to represent 0-9 as char
                }
                else{
               EmsgHEX[++Emsgi]=(char)(sss+87); // to represent a-f as char
                }
    }
}
/* */

 /* Converting Binary Matrix to char
  * Logic - (7th bit *128)+(6th bit *64)+(5th bit *32)+ .... +(1st bit*2)+(0th bit*1)
  * 8 rows i.e. one character per row
  * For each char, 8 columns i.e. 1 bit per column
  */
```

```
          for(i=0;i<8;i++){
           k=128;  // as 2^7 = 128
           d=0;
           for(j=0;j<8;j++){
              d+= inv[i][j]*k;
              k=k/2;
           }
           // Ascii to char
           final[++encryptedMsgIndex]=(char)d;

       }
    }
  final[++encryptedMsgIndex]='\0';


  cout<<"\nEncyption Key HEX  : 133457799bbcdff1";
  cout<<"\nPlain  Text   HEX  : "<<OmsgHEX;
  cout<<"\nCipher Text   HEX  : "<<EmsgHEX;

  return final;
}

/*
Handles the complete key generation process for all cycles.
*/
void Des::keygen()
{
            /* PC-1 generated & applied.
            Converts 64 bit key to 56 bit key
            */PermChoice1();
```

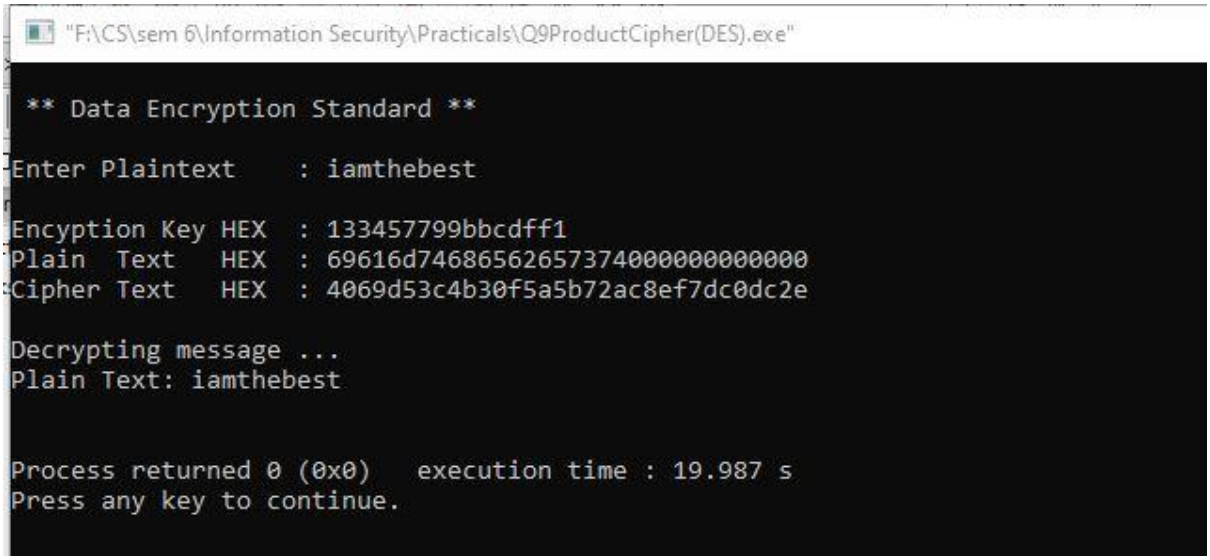
```
int i,j;
// splitting into Left (Ck) and Right (Dk) Halves of 28-28 each
for(i=0;i<28;i++){
	ck[i]=pc1[i];
}
int k=0;
for(i=28;i<56;i++){
	dk[k]=pc1[i];
	k++;
}

int noshift=0,round;
for(round=1;round<=16;round++)
{
	// no of Left shifts required at each Cycle
	if(round==1||round==2||round==9||round==16)
		noshift=1;
	else
		noshift=2;

	// shift key by required
	while(noshift--){
		int t;
		t=ck[0];
		for(i=0;i<28;i++)
		ck[i]=ck[i+1];
		ck[27]=t;

		t=dk[0];
		for(i=0;i<28;i++)
		dk[i]=dk[i+1];
```

```
                dk[27]=t;
        }

        /* applied permutation choice
        Converts 56 bit key to 48 bit key
        */PermChoice2();

        // Hold Kn of this cycle in Keyi array, round-1 to manage index
        for(i=0;i<48;i++)
                keyi[round-1][i]=z[i];
    }
}

/*
Decryption is the same as Encryption
but subkeys are used in opposite order.
*/
char * Des::Decrypt(char *Text1){

 int i,a1,j,nB,m,iB,k,K,B[8],n,t,d,round;

 // generate all required keys - same as encryption
 keygen();

 // making copy for safety
 char *Text=new char[1000];
 strcpy(Text,Text1);

 unsigned char ch;

 // Unlike encryption, message will always be multiple of 8, as it comes to us after encryption
```

```
i=strlen(Text);
        int mc=0;

for(iB=0,nB=0,m=0;m<(strlen(Text)/8);m++)
{
         for(iB=0,i=0;i<8;i++,nB++)
                  {
                   ch=Text[nB];
                   n=(int)ch ;
                   for(K=7;n>=1;K--)
                   {
                    B[K]=n%2;
                    n/=2;
                   } for(;K>=0;K--) B[K]=0;
                  for(K=0;K<8;K++,iB++) total[iB]=B[K];
                  }

        IP();
        for(i=0;i<64;i++) total[i]=ip[i];
        for(i=0;i<32;i++) left[i]=total[i];
        for(;i<64;i++) right[i-32]=total[i];

        for(round=1;round<=16;round++)
        {
                  Expansion();
                  xor_oneD(round);
                  substitution();
                  permutation();
                  xor_two();
                  for(i=0;i<32;i++) left[i]=right[i];
                  for(i=0;i<32;i++) right[i]=xor2[i];
```

```
            }

            for(i=0;i<32;i++) temp[i]=right[i];
            for(;i<64;i++) temp[i]=left[i-32];

            inverseIP();
            for(i=0;i<8;i++)
            {
                        k=128;   d=0;
                         for(j=0;j<8;j++)
             {
                         d=d+inv[i][j]*k;
                         k=k/2;
             }
                        final[mc++]=(char)d;
            }
 }
 final[mc]='\0';

 char *final1=new char[1000];
 for(i=0,j=strlen(Text);i<strlen(Text);i++,j++)
             final1[i]=final[j]; final1[i]='\0';
 return(final);
}


int main()
{
 Des d1,d2;
 char *str=new char[1000];
 char *str1=new char[1000];
```

```
cout<<"\n ** Data Encryption Standard **\n";
cout<<"\nEnter Plaintext    : ";  cin>>str;

str1=d1.Encrypt(str);
cout<<"\n";

cout<<"\nDecrypting message ... ";

        cout<<"\nPlain Text: "<<d2.Decrypt(str1)<<"\n\n";

        //cout<<"\nPlain Text: "<<str<<endl;
        // cout<<"\nCipher Text  : \n"<<str1<<endl;
return 0;
}
```



10.Illustrate the Ciphertext only and Known Plaintext attacks.

Cipher Text Only

In cryptography, a ciphertext-only attack (COA) or known ciphertext attack is an attack model for cryptanalysis where the attacker is assumed to have access only to a set of ciphertexts.The attack is completely successful if the corresponding plaintexts can be deduced, or even better, the key. The ability to obtain any information at all about the underlying plaintext is still considered a success. For example, if an adversary is sending ciphertext continuously to maintain traffic-flow security, it would

be very useful to be able to distinguish real messages from nulls. Even making an informed guess of the existence of real messages would facilitate traffic analysis. Every modern cipher attempts to provide protection against ciphertext only attacks. The vetting process for a new cipher design standard usually takes many years and includes exhaustive testing of large quantities of ciphertext for any statistical departure from random noise. Encryption Standard process. Also, the field of steganography evolved, in part, to develop methods like mimic functions that allow one piece of data to adopt the statistical profile of another. Nonetheless poor cipher usage or reliance on home-grown proprietary algorithms that have notbeen subject to thorough scrutiny has resulted in many computer-age encryption systems that are still subject to ciphertext-only attack.

Known Plaintext Attack

The known-plaintext attack (KPA) or crib is an attack model for cryptanalysis where the attacker has samples of both the plaintext and its encrypted version (ciphertext), and is at liberty to make use of them to reveal further secret information such as secret keys and code books. The term "crib" originated at Bletchley Park, the British World War II decryption operation. Classical ciphers are typically vulnerable to known-plaintext attack. For example, a Caesar cipher can be solved using a single letter of corresponding plaintext and ciphertext to decrypt entirely. A general monoalphabetic substitution cipher needs several character pairs and some guessing if there are fewer than 26 distinct pairs. Modern ciphers such as Advanced Encryption Standard are not susceptible to known-plaintext attacks.

11.Implement a stream cipher technique

```
#include<stdio.h>

#include<string.h>


void cipher(int i, int c);

int findMin();

void makeArray(int, int);


char arr[22][22], darr[22][22], emessage[111], retmessage[111], key[55];

char temp[55], temp2[55];

int k = 0;


int main()

{

   char *message;
```

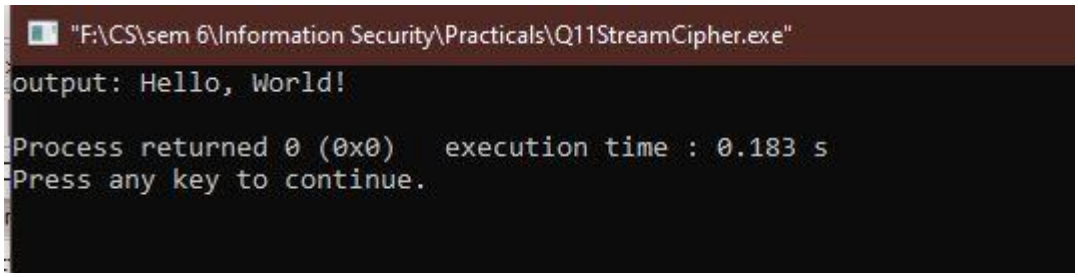
```
int i, j, klen, emlen, flag = 0;
int r, c, index, rows;

printf("Enter the key\n");
fflush(stdin);
gets(key);

printf("\nEnter message to be ciphered\n");
fflush(stdin);
gets(message);

strcpy(temp, key);
klen = strlen(key);

k = 0;
for (i = 0;; i++)
{
   if (flag == 1)
      break;

   for (j = 0; key[j] != NULL; j++)
   {
      if (message[k] == NULL)
      {
         flag = 1;
         arr[i][j] = '-';
      }
      else
      {
         arr[i][j] = message[k++];
      }
```

```
    }
}
r = i;
c = j;

for (i = 0; i < r; i++)
{
    for (j = 0; j < c; j++)
    {
        printf("%c ", arr[i][j]);
    }
    printf("\n");
}

k = 0;

for (i = 0; i < klen; i++)
{
    index = findMin();
    cipher(index, r);
}

emessage[k] = '\0';
printf("\nEncrypted message is\n");
for (i = 0; emessage[i] != NULL; i++)
    printf("%c", emessage[i]);

printf("\n\n");
//deciphering

emlen = strlen(emessage);
```

```
//emlen is length of encrypted message

strcpy(temp, key);

rows = emlen / klen;
//rows is no of row of the array to made from ciphered message

j = 0;

for (i = 0, k = 1; emessage[i] != NULL; i++, k++)
{
   //printf("\nEmlen=%d",emlen);
   temp2[j++] = emessage[i];
   if ((k % rows) == 0)
   {
      temp2[j] = '\0';
      index = findMin();
      makeArray(index, rows);
      j = 0;
   }
}

printf("\nArray Retrieved is\n");

k = 0;
for (i = 0; i < r; i++)
{
   for (j = 0; j < c; j++)
   {
      printf("%c ", darr[i][j]);
      //retrieving message
```

```
            retmessage[k++] = darr[i][j];

        }
        printf("\n");
    }
    retmessage[k] = '\0';

    printf("\nMessage retrieved is\n");

    for (i = 0; retmessage[i] != NULL; i++)
        printf("%c", retmessage[i]);

    return (0);
}

void cipher(int i, int r)
{
    int j;
    for (j = 0; j < r; j++)
    {
        {
            emessage[k++] = arr[j][i];
        }
    }
    // emessage[k]='\0';
}

void makeArray(int col, int row)
{
    int i, j;
```

```
    for (i = 0; i < row; i++)
    {
        darr[i][col] = temp2[i];
    }
}

int findMin()
{
    int i, j, min, index;

    min = temp[0];
    index = 0;
    for (j = 0; temp[j] != NULL; j++)
    {
        if (temp[j] < min)
        {
            min = temp[j];
            index = j;
        }
    }

    temp[index] = 123;
    return (index);
}
```



END