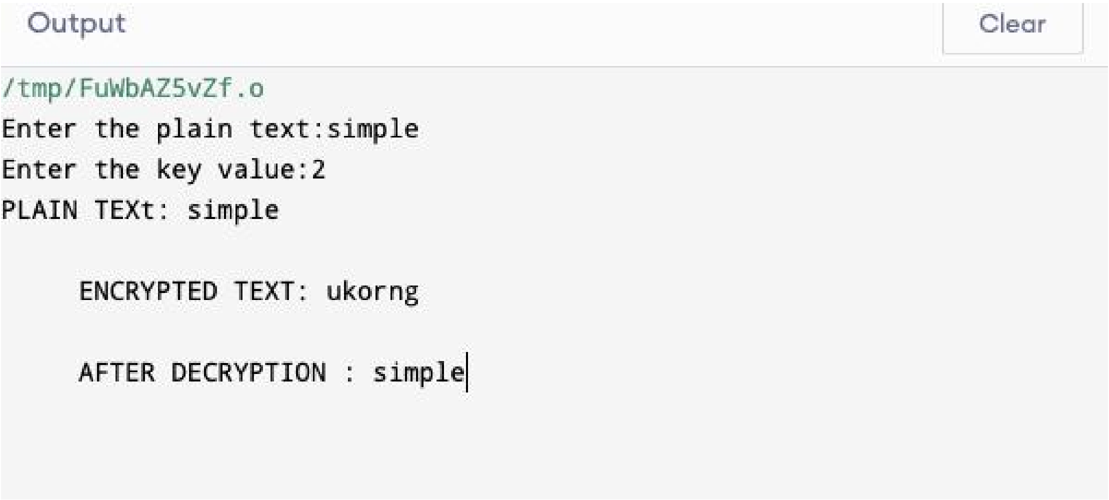
PROGRAM: (Caesar Cipher)

```
#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();
}
```



```
/tmp/FuWbAZ5vZf.o
Enter the plain text:simple
Enter the key value:2
PLAIN TEXt: simple

     ENCRYPTED TEXT: ukorng

     AFTER DECRYPTION : simple
```

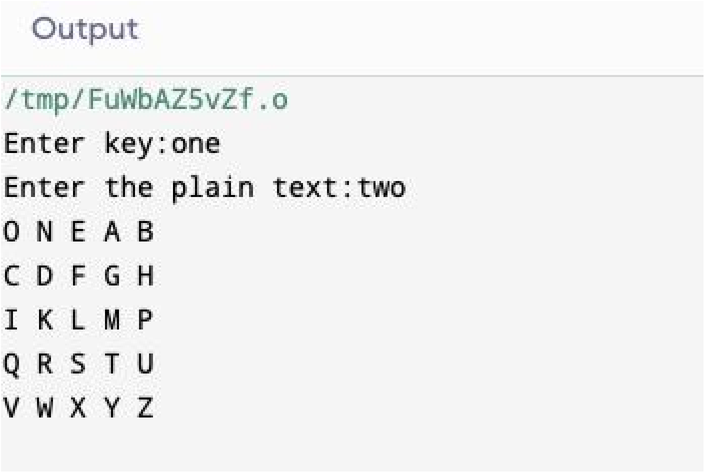
PROGRAM: (Playfair Cipher)

```
#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
{}
//getch();
}
if(str[i+1]=='J')str[i+1]='I';
if(str[i]==str[i+1])
playfair(str[i],'X',key);
```

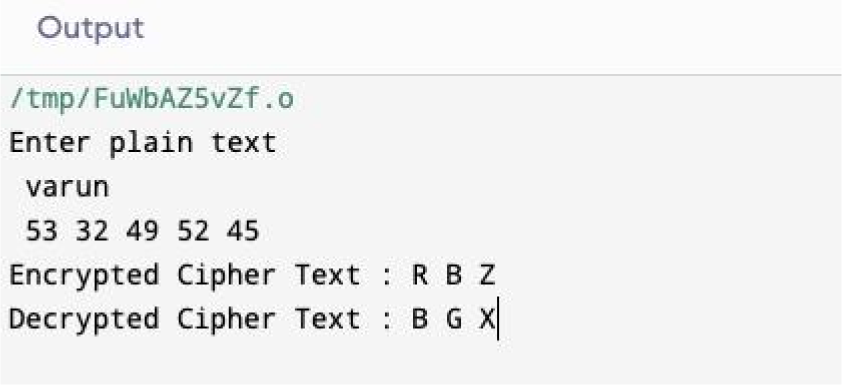
```
else
{
playfair(str[i],str[i+1],key);i++;
}}
```



```
/tmp/FuWbAZ5vZf.o
Enter key:one
Enter the plain text:two
O N E A B
C D F G H
I K L M P
Q R S T U
V W X Y Z
```

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;
}
```



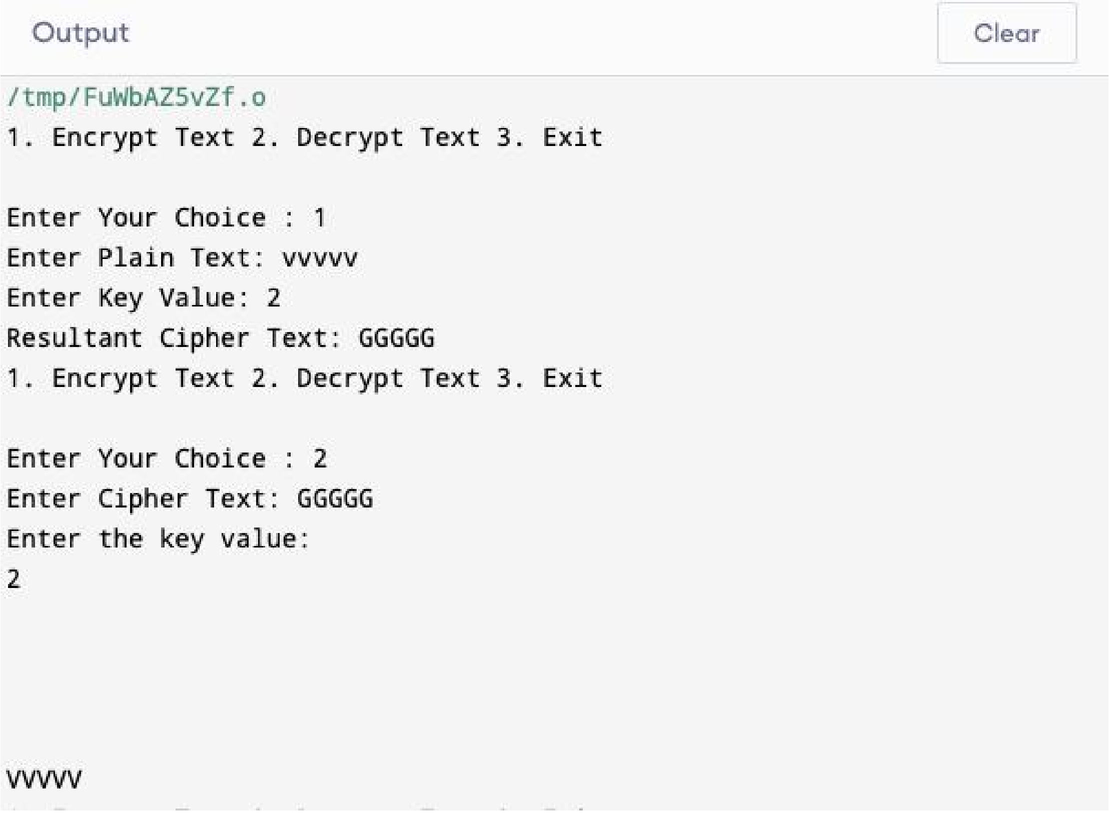
```
/tmp/FuWbAZ5vZf.o
Enter plain text
 varun
 53 32 49 52 45
Encrypted Cipher Text : R B Z
Decrypted Cipher Text : B G X
```

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));
}}
```

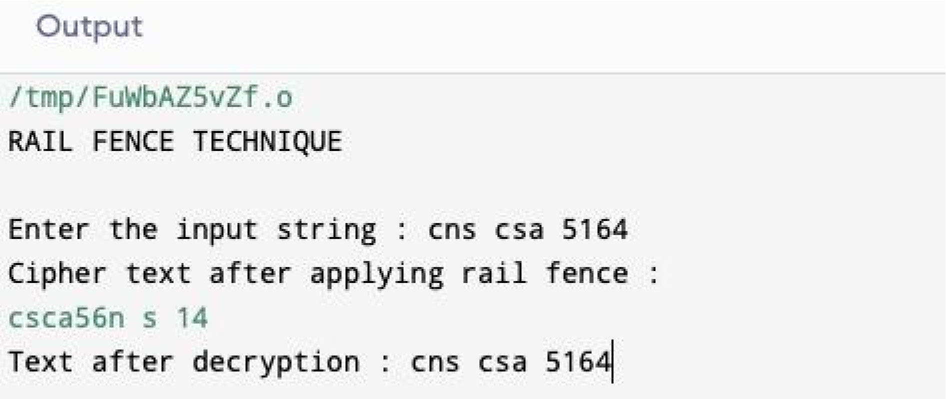


```
/tmp/FuWbAZ5vZf.o
1. Encrypt Text 2. Decrypt Text 3. Exit

Enter Your Choice : 1
Enter Plain Text: vvvvv
Enter Key Value: 2
Resultant Cipher Text: GGGGG
1. Encrypt Text 2. Decrypt Text 3. Exit

Enter Your Choice : 2
Enter Cipher Text: GGGGG
Enter the key value:
2




vvvvv
```

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();
}
```



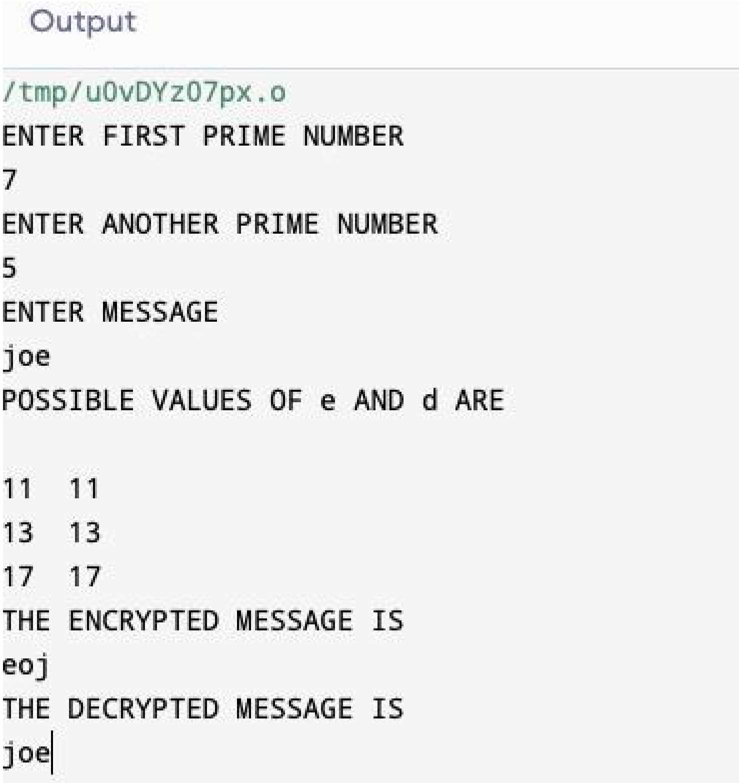
```
/tmp/FuWbAZ5vZf.o
RAIL FENCE TECHNIQUE

Enter the input string : cns csa 5164
Cipher text after applying rail fence :
csca56n s 14
Text after decryption : cns csa 5164
```

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;
```

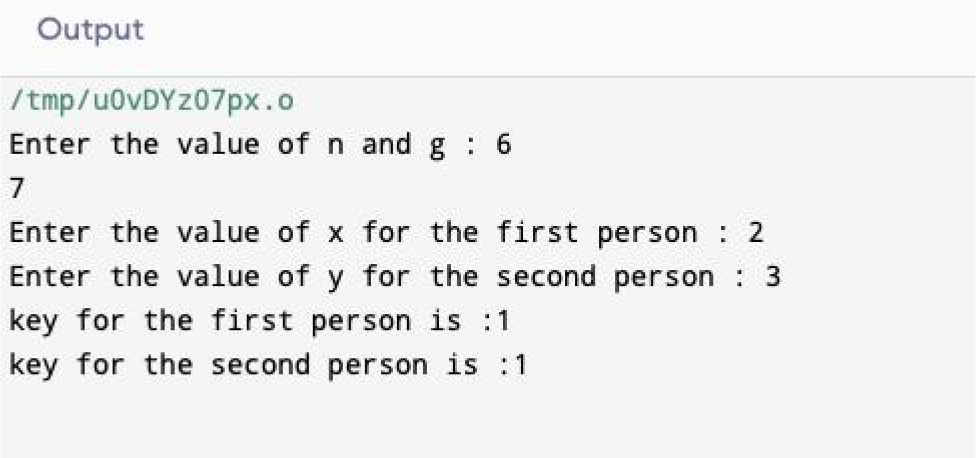
```
j=sqrt(pr);
for(i=2;i<=j;i++)
{
if(pr%i==0)
return 0;
}
return 1;
}
void ce()
{
int k;
k=0;
for(i=2;i<t;i++)
{
if(t%i==0)
continue;
flag=prime(i);
if(flag==1&&i!=p&&i!=q)
{
e[k]=i;
flag=cd(e[k]);
if(flag>0)
{
d[k]=flag;
k++;
}
if(k==99)
break;
} } }
long int cd(long int x)
{
long int k=1;
while(1)
{
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]);
}
```



```
/tmp/u0vDYz07px.o
ENTER FIRST PRIME NUMBER
7
ENTER ANOTHER PRIME NUMBER
5
ENTER MESSAGE
joe
POSSIBLE VALUES OF e AND d ARE

11  11
13  13
17  17
THE ENCRYPTED MESSAGE IS
eoj
THE DECRYPTED MESSAGE IS
joe
```

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

```
/tmp/u0vDYz07px.o
Enter the value of n and g : 6
7
Enter the value of x for the first person : 2
Enter the value of y for the second person : 3
key for the first person is :1
key for the second person is :1
```

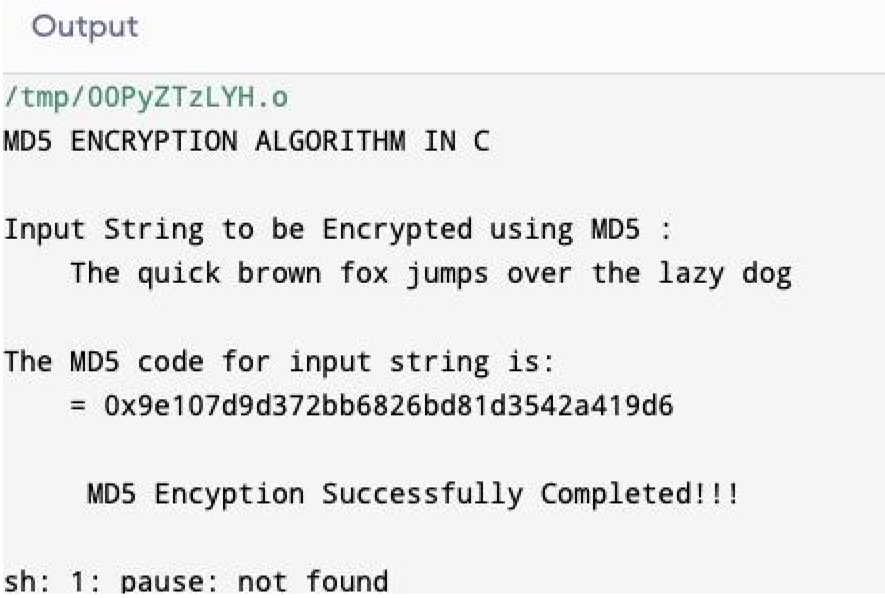
Program(md5):

```
#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;}
int main()
{
int j,k;
const char *msg = "The quick brown fox jumps over the lazy dog";
unsigned *d = md5(msg, strlen(msg));
```

```
MD5union u;
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();
}
```

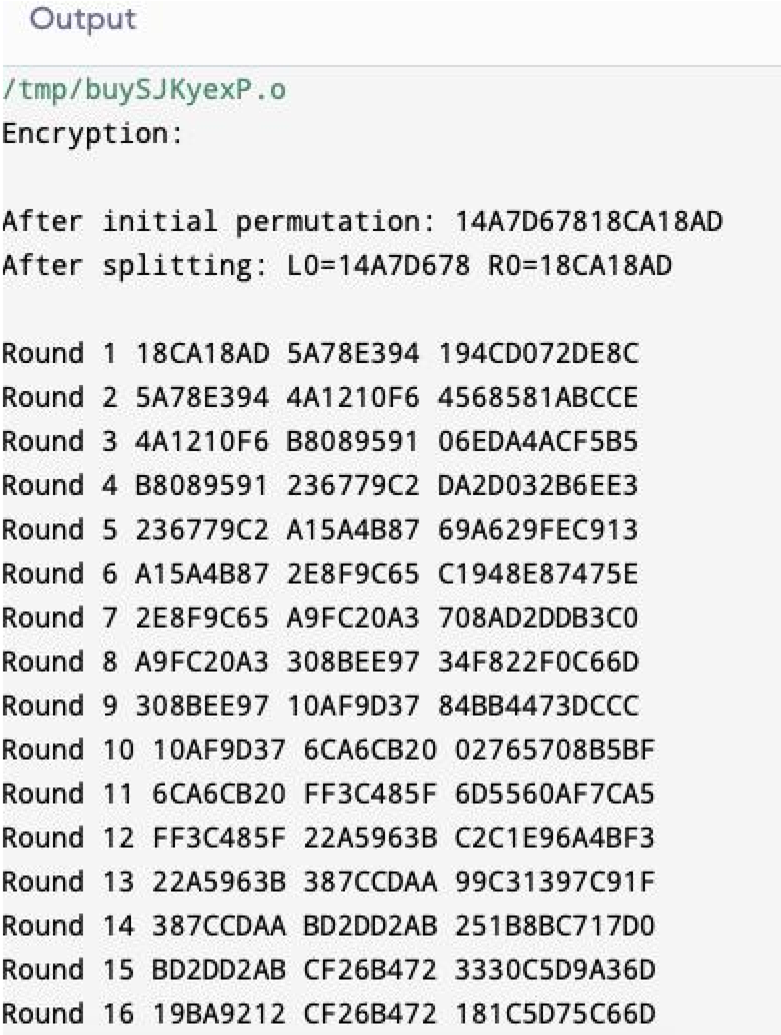


```
/tmp/OOPyZTzLYH.o
MD5 ENCRYPTION ALGORITHM IN C

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

The MD5 code for input string is:
    = 0x9e107d9d372bb6826bd81d3542a419d6


     MD5 Encyption Successfully Completed!!!

sh: 1: pause: not found
```

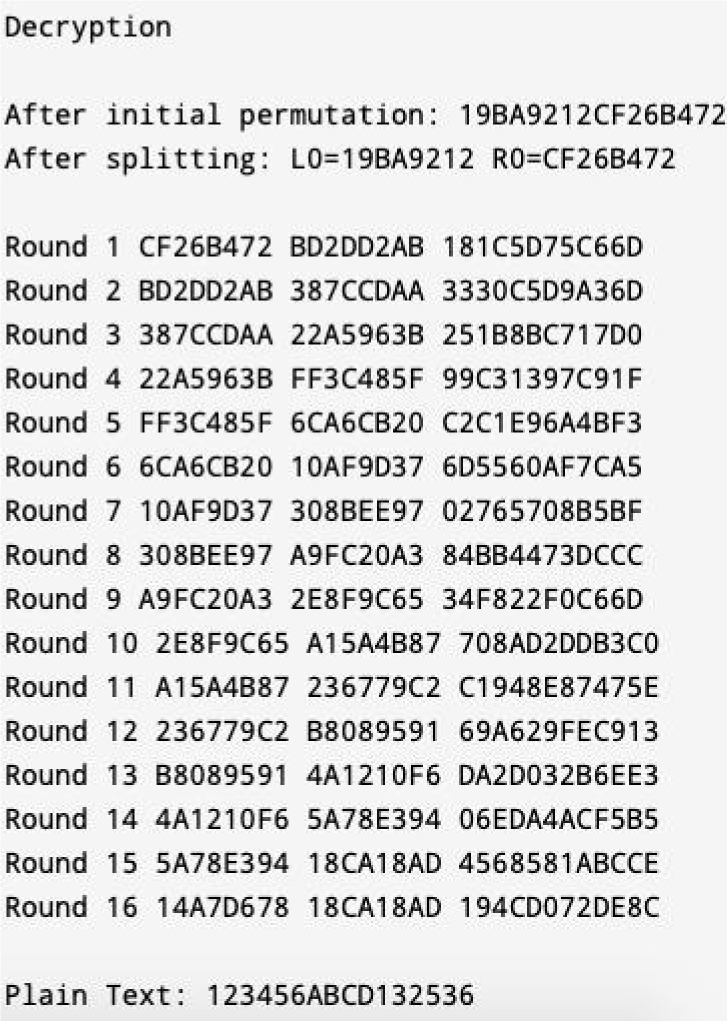
Program(des)

```
#include <bits/stdc++.h>
using namespace std;
string hex2bin(string s)
{
    // hexadecimal to binary conversion
    unordered_map<char, string> mp;
    mp['0'] = "0000";
    mp['1'] = "0001";
    mp['2'] = "0010";
    mp['3'] = "0011";
    mp['4'] = "0100";
    mp['5'] = "0101";
    mp['6'] = "0110";
    mp['7'] = "0111";
    mp['8'] = "1000";
    mp['9'] = "1001";
    mp['A'] = "1010";
    mp['B'] = "1011";
    mp['C'] = "1100";
    mp['D'] = "1101";
    mp['E'] = "1110";
    mp['F'] = "1111";
    string bin = "";
    for (int i = 0; i < s.size(); i++) {
        bin += mp[s[i]];
    }
    return bin;
}
string bin2hex(string s)
{
    // binary to hexadecimal conversion
    unordered_map<string, string> mp;
    mp["0000"] = "0";
    mp["0001"] = "1";
    mp["0010"] = "2";
    mp["0011"] = "3";
    mp["0100"] = "4";
    mp["0101"] = "5";
    mp["0110"] = "6";
    mp["0111"] = "7";
    mp["1000"] = "8";
    mp["1001"] = "9";
    mp["1010"] = "A";
    mp["1011"] = "B";
    mp["1100"] = "C";
    mp["1101"] = "D";
    mp["1110"] = "E";
    mp["1111"] = "F";
    string hex = "";
    for (int i = 0; i < s.length(); i += 4) {
        string ch = "";
        ch += s[i];
        ch += s[i + 1];
        ch += s[i + 2];
        ch += s[i + 3];
        hex += mp[ch];
    }
    return hex;
}

string permute(string k, int* arr, int n)
{
    string per = "";
    for (int i = 0; i < n; i++) {
        per += k[arr[i] - 1];
    }
    return per;
}

string shift_left(string k, int shifts)
{
    string s = "";
    for (int i = 0; i < shifts; i++) {
        for (int j = 1; j < 28; j++) {
            s += k[j];
        }
        s += k[0];
        k = s;
        s = "";
    }
    return k;
}

string xor_(string a, string b)
{
    string ans = "";
    for (int i = 0; i < a.size(); i++) {
        if (a[i] == b[i]) {
            ans += "0";
        }
        else {
            ans += "1";
        }
    }
    return ans;
}
string encrypt(string pt, vector<string> rkb,
            vector<string> rk)
```

```
{
    // Hexadecimal to binary
    pt = hex2bin(pt);

    // Initial Permutation Table
    int initial_perm[64]
        = { 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44,
            36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22,
            14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57,
            49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35,
            27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13,
            5, 63, 55, 47, 39, 31, 23, 15, 7 };
    // Initial Permutation
    pt = permute(pt, initial_perm, 64);
    cout << "After initial permutation: " << bin2hex(pt)
        << endl;

    // Splitting
    string left = pt.substr(0, 32);
    string right = pt.substr(32, 32);
    cout << "After splitting: L0=" << bin2hex(left)
        << " R0=" << bin2hex(right) << endl;

    // Expansion D-box Table
    int exp_d[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 };

    // S-box Table
    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 }
};

// Straight Permutation Table
int per[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 };

cout << endl;
for (int i = 0; i < 16; i++) {
    // Expansion D-box
    string right_expanded = permute(right, exp_d, 48);

    // XOR RoundKey[i] and right_expanded
    string x = xor_(rkb[i], right_expanded);

    // S-boxes
    string op = "";
    for (int i = 0; i < 8; i++) {
        int row = 2 * int(x[i * 6] - '0')
                + int(x[i * 6 + 5] - '0');
        int col = 8 * int(x[i * 6 + 1] - '0')
                + 4 * int(x[i * 6 + 2] - '0')
                + 2 * int(x[i * 6 + 3] - '0')
                + int(x[i * 6 + 4] - '0');
        int val = s[i][row][col];
        op += char(val / 8 + '0');
        val = val % 8;
        op += char(val / 4 + '0');
        val = val % 4;
        op += char(val / 2 + '0');
        val = val % 2;
        op += char(val + '0');
    }
    // Straight D-box
    op = permute(op, per, 32);

    // XOR left and op
    x = xor_(op, left);

    left = x;

    // Swapper
    if (i != 15) {
        swap(left, right);
    }
    cout << "Round " << i + 1 << " " << bin2hex(left)
        << " " << bin2hex(right) << " " << rk[i]
        << endl;
}

// Combination
string combine = left + right;

// Final Permutation Table
int final_perm[64]
    = { 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47,
        15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22,
            62, 30, 37, 5, 45, 13, 53, 21, 61, 29, 36,
            4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11,
            51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58,
            26, 33, 1, 41, 9, 49, 17, 57, 25 };

    // Final Permutation
    string cipher
        = bin2hex(permute(combine, final_perm, 64));
    return cipher;
}

// Driver code
int main()
{
    // pt is plain text
    string pt, key;
    /*cout<<"Enter plain text(in hexadecimal): ";
    cin>>pt;
    cout<<"Enter key(in hexadecimal): ";
    cin>>key;*/

    pt = "123456ABCD132536";
    key = "AABB09182736CCDD";
    // Key Generation

    // Hex to binary
    key = hex2bin(key);

    // Parity bit drop table
    int keyp[56]
        = { 57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34,
            26, 18, 10, 2, 59, 51, 43, 35, 27, 19, 11, 3,
            60, 52, 44, 36, 63, 55, 47, 39, 31, 23, 15, 7,
            62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37,
            29, 21, 13, 5, 28, 20, 12, 4 };

    // getting 56 bit key from 64 bit using the parity bits
    key = permute(key, keyp, 56); // key without parity

    // Number of bit shifts
    int shift_table[16] = { 1, 1, 2, 2, 2, 2, 2, 2,
                            1, 2, 2, 2, 2, 2, 2, 1 };

    // Key- Compression Table
    int key_comp[48] = { 14, 17, 11, 24, 1, 5, 3, 28,
                         15, 6, 21, 10, 23, 19, 12, 4,
                         26, 8, 16, 7, 27, 20, 13, 2,
                         41, 52, 31, 37, 47, 55, 30, 40,
                         51, 45, 33, 48, 44, 49, 39, 56,
                         34, 53, 46, 42, 50, 36, 29, 32 };

    // Splitting
    string left = key.substr(0, 28);
    string right = key.substr(28, 28);

    vector<string> rkb; // rkb for RoundKeys in binary
    vector<string> rk; // rk for RoundKeys in hexadecimal
    for (int i = 0; i < 16; i++) {
        // Shifting
        left = shift_left(left, shift_table[i]);
        right = shift_left(right, shift_table[i]);

        // Combining
        string combine = left + right;

        // Key Compression
        string RoundKey = permute(combine, key_comp, 48);

        rkb.push_back(RoundKey);
        rk.push_back(bin2hex(RoundKey));
    }

    cout << "\nEncryption:\n\n";
    string cipher = encrypt(pt, rkb, rk);
    cout << "\nCipher Text: " << cipher << endl;

    cout << "\nDecryption\n\n";
    reverse(rkb.begin(), rkb.end());
    reverse(rk.begin(), rk.end());
    string text = encrypt(cipher, rkb, rk);
    cout << "\nPlain Text: " << text << endl;
}
```



```
/tmp/buySJKyexP.o
Encryption:

After initial permutation: 14A7D67818CA18AD
After splitting: L0=14A7D678 R0=18CA18AD

Round 1 18CA18AD 5A78E394 194CD072DE8C
Round 2 5A78E394 4A1210F6 4568581ABCCE
Round 3 4A1210F6 B8089591 06EDA4ACF5B5
Round 4 B8089591 236779C2 DA2D032B6EE3
Round 5 236779C2 A15A4B87 69A629FEC913
Round 6 A15A4B87 2E8F9C65 C1948E87475E
Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0
Round 8 A9FC20A3 308BEE97 34F822F0C66D
Round 9 308BEE97 10AF9D37 84BB4473DCCC
Round 10 10AF9D37 6CA6CB20 02765708B5BF
Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5
Round 12 FF3C485F 22A5963B C2C1E96A4BF3
Round 13 22A5963B 387CCDAA 99C31397C91F
Round 14 387CCDAA BD2DD2AB 251B8BC717D0
Round 15 BD2DD2AB CF26B472 3330C5D9A36D
Round 16 19BA9212 CF26B472 181C5D75C66D
```



```
Decryption

After initial permutation: 19BA9212CF26B472
After splitting: L0=19BA9212 R0=CF26B472

Round 1 CF26B472 BD2DD2AB 181C5D75C66D
Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D
Round 3 387CCDAA 22A5963B 251B8BC717D0
Round 4 22A5963B FF3C485F 99C31397C91F
Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3
Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5
Round 7 10AF9D37 308BEE97 02765708B5BF
Round 8 308BEE97 A9FC20A3 84BB4473DCCC
Round 9 A9FC20A3 2E8F9C65 34F822F0C66D
Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0
Round 11 A15A4B87 236779C2 C1948E87475E
Round 12 236779C2 B8089591 69A629FEC913
Round 13 B8089591 4A1210F6 DA2D032B6EE3
Round 14 4A1210F6 5A78E394 06EDA4ACF5B5
Round 15 5A78E394 18CA18AD 4568581ABCCE
Round 16 14A7D678 18CA18AD 194CD072DE8C

Plain Text: 123456ABCD132536
```

Program(dsa)

```
#include <stdio.h>
#include <string.h>

static void display(int intArray[], int length){
int i=0;
printf("Array : [");
for(i = 0; i < length; i++) {
/* display value of element at index i. */
printf(" %d ", intArray[i]);
}
printf(" ]\n ");
}
```

```
int main() {
int i = 0;
```

```c
/* Declare an array */
int intArray[8];
```

```c
// initialize elements of array n to 0
for ( i = 0; i < 8; i++ ) {
intArray[ i ] = 0; // set elements to default value of 0;
}
printf("Array with default data.");
```

```c
/* Display elements of an array.*/
display(intArray,8);
```

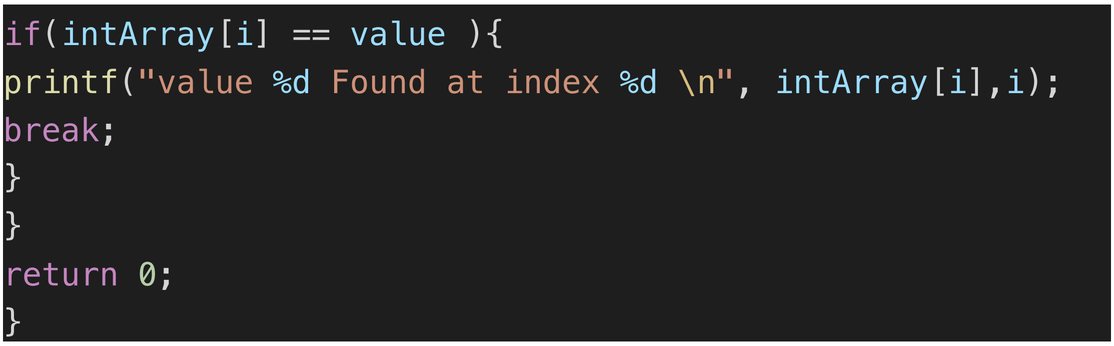
```c
/* Operation : Insertion
Add elements in the array */
for(i = 0; i < 8; i++) {
/* place value of i at index i. */
printf("Adding %d at index %d\n",i,i);
intArray[i] = i;
}
printf("\n");
printf("Array after adding data. ");
display(intArray,8);
```

```c
/* Operation : Insertion
Element at any location can be updated directly */
int index = 5;
intArray[index] = 10;
```

```c
printf("Array after updating element at index %d.\n",index);
display(intArray,8);
```

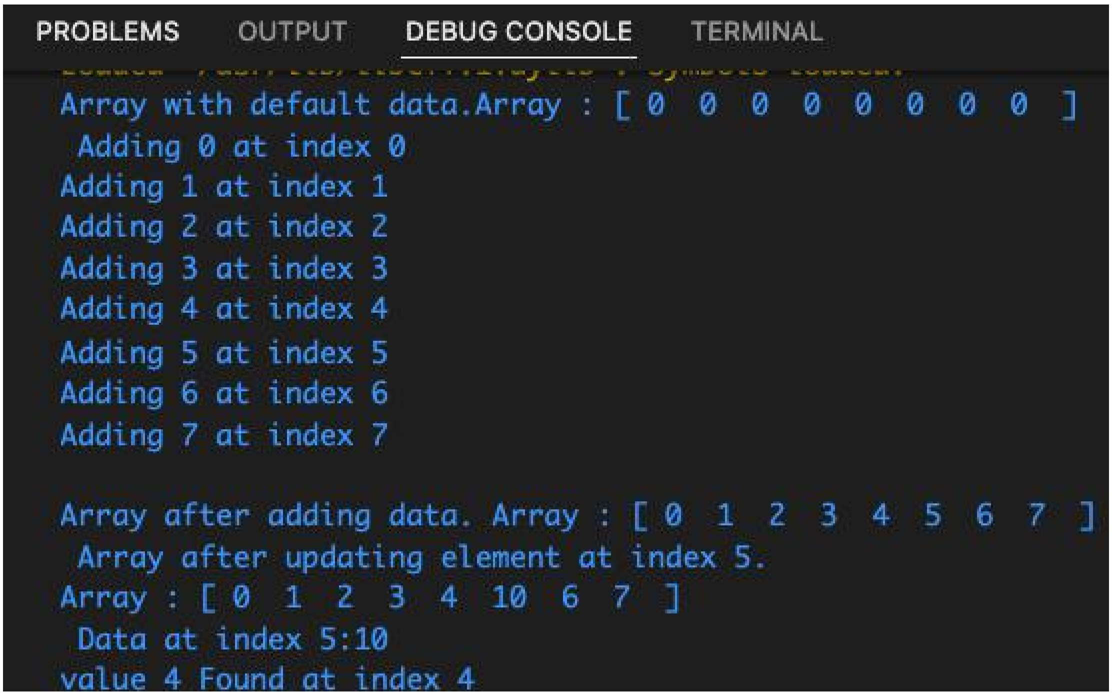
```c
/* Operation : Search using index
Search an element using index.*/
printf("Data at index %d:%d\n" ,index,intArray[index]);
```

```c
/* Operation : Search using value
Search an element using value.*/
int value = 4;
for(i = 0; i < 8; i++) {
```



```
if(intArray[i] == value ){
printf("value %d Found at index %d \n", intArray[i],i);
break;
}
}
return 0;
}
```

Output:

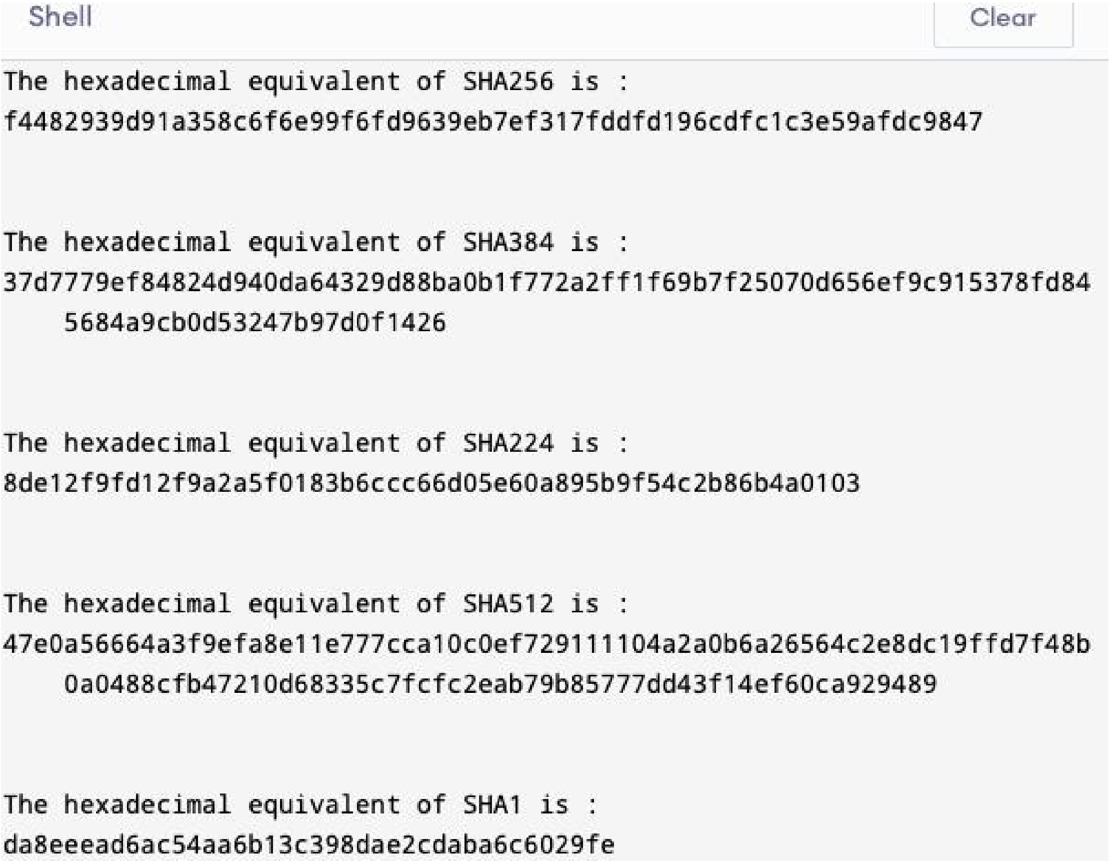


```
Array with default data.Array : [ 0  0  0  0  0  0  0  0  ]
 Adding 0 at index 0
Adding 1 at index 1
Adding 2 at index 2
Adding 3 at index 3
Adding 4 at index 4
Adding 5 at index 5
Adding 6 at index 6
Adding 7 at index 7

Array after adding data. Array : [ 0  1  2  3  4  5  6  7  ]
 Array after updating element at index 5.
Array : [ 0  1  2  3  4  10  6  7  ]
 Data at index 5:10
value 4 Found at index 4
```

Sha program (python):

```
import hashlib


str = "varun"


result = hashlib.sha256(str.encode())


print("The hexadecimal equivalent of SHA256 is : ")
print(result.hexdigest())

print ("\r")
```

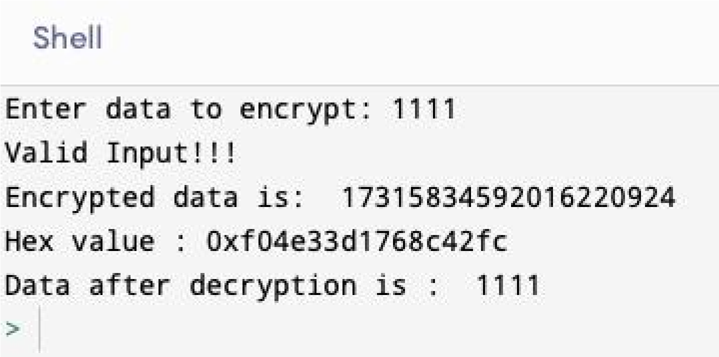
```
str = "varun"


result = hashlib.sha384(str.encode())


print("The hexadecimal equivalent of SHA384 is : ")
print(result.hexdigest())

print ("\r")


str = "varun"


result = hashlib.sha224(str.encode())


print("The hexadecimal equivalent of SHA224 is : ")
print(result.hexdigest())

print ("\r")


str = "varun"


result = hashlib.sha512(str.encode())


print("The hexadecimal equivalent of SHA512 is : ")
print(result.hexdigest())

print ("\r")


str = "varun"


result = hashlib.sha1(str.encode())


print("The hexadecimal equivalent of SHA1 is : ")
print(result.hexdigest())
```



```
The hexadecimal equivalent of SHA256 is :
f4482939d91a358c6f6e99f6fd9639eb7ef317fddfd196cdfc1c3e59afdc9847


The hexadecimal equivalent of SHA384 is :
37d7779ef84824d940da64329d88ba0b1f772a2ff1f69b7f25070d656ef9c915378fd84
    5684a9cb0d53247b97d0f1426


The hexadecimal equivalent of SHA224 is :
8de12f9fd12f9a2a5f0183b6ccc66d05e60a895b9f54c2b86b4a0103


The hexadecimal equivalent of SHA512 is :
47e0a56664a3f9efa8e11e777cca10c0ef729111104a2a0b6a26564c2e8dc19ffd7f48b
    0a0488cfb47210d68335c7fcfc2eab79b85777dd43f14ef60ca929489


The hexadecimal equivalent of SHA1 is :
da8eeead6ac54aa6b13c398dae2cdaba6c6029fe
```

Blowfish(python):

```
p = [
        0x243F6A88, 0x85A308D3, 0x13198A2E, 0x03707344,
        0xA4093822, 0x299F31D0, 0x082EFA98, 0xEC4E6C89,
        0x452821E6, 0x38D01377, 0xBE5466CF, 0x34E90C6C,
        0xC0AC29B7, 0xC97C50DD, 0x3F84D5B5, 0xB5470917,
        0x9216D5D9, 0x8979FB1B
  ]

s = [
        [
             0xD1310BA6, 0x98DFB5AC, 0x2FFD72DB, 0xD01ADFB7,
             0xB8E1AFED, 0x6A267E96, 0xBA7C9045, 0xF12C7F99,
             0x24A19947, 0xB3916CF7, 0x0801F2E2, 0x858EFC16,
             0x636920D8, 0x71574E69, 0xA458FEA3, 0xF4933D7E,
             0x0D95748F, 0x728EB658, 0x718BCD58, 0x82154AEE,
             0x7B54A41D, 0xC25A59B5, 0x9C30D539, 0x2AF26013,
             0xC5D1B023, 0x286085F0, 0xCA417918, 0xB8DB38EF,
             0x8E79DCB0, 0x603A180E, 0x6C9E0E8B, 0xB01E8A3E,
             0xD71577C1, 0xBD314B27, 0x78AF2FDA, 0x55605C60,
0xE65525F3, 0xAA55AB94, 0x57489862, 0x63E81440,
0x55CA396A, 0x2AAB10B6, 0xB4CC5C34, 0x1141E8CE,
0xA15486AF, 0x7C72E993, 0xB3EE1411, 0x636FBC2A,
0x2BA9C55D, 0x741831F6, 0xCE5C3E16, 0x9B87931E,
0xAFD6BA33, 0x6C24CF5C, 0x7A325381, 0x28958677,
0x3B8F4898, 0x6B4BB9AF, 0xC4BFE81B, 0x66282193,
0x61D809CC, 0xFB21A991, 0x487CAC60, 0x5DEC8032,
0xEF845D5D, 0xE98575B1, 0xDC262302, 0xEB651B88,
0x23893E81, 0xD396ACC5, 0x0F6D6FF3, 0x83F44239,
0x2E0B4482, 0xA4842004, 0x69C8F04A, 0x9E1F9B5E,
0x21C66842, 0xF6E96C9A, 0x670C9C61, 0xABD388F0,
0x6A51A0D2, 0xD8542F68, 0x960FA728, 0xAB5133A3,
0x6EEF0B6C, 0x137A3BE4, 0xBA3BF050, 0x7EFB2A98,
0xA1F1651D, 0x39AF0176, 0x66CA593E, 0x82430E88,
0x8CEE8619, 0x456F9FB4, 0x7D84A5C3, 0x3B8B5EBE,
0xE06F75D8, 0x85C12073, 0x401A449F, 0x56C16AA6,
0x4ED3AA62, 0x363F7706, 0x1BFEDF72, 0x429B023D,
0x37D0D724, 0xD00A1248, 0xDB0FEAD3, 0x49F1C09B,
0x075372C9, 0x80991B7B, 0x25D479D8, 0xF6E8DEF7,
0xE3FE501A, 0xB6794C3B, 0x976CE0BD, 0x04C006BA,
0xC1A94FB6, 0x409F60C4, 0x5E5C9EC2, 0x196A2463,
0x68FB6FAF, 0x3E6C53B5, 0x1339B2EB, 0x3B52EC6F,
0x6DFC511F, 0x9B30952C, 0xCC814544, 0xAF5EBD09,
0xBEE3D004, 0xDE334AFD, 0x660F2807, 0x192E4BB3,
0xC0CBA857, 0x45C8740F, 0xD20B5F39, 0xB9D3FBDB,
0x5579C0BD, 0x1A60320A, 0xD6A100C6, 0x402C7279,
0x679F25FE, 0xFB1FA3CC, 0x8EA5E9F8, 0xDB3222F8,
0x3C7516DF, 0xFD616B15, 0x2F501EC8, 0xAD0552AB,
0x323DB5FA, 0xFD238760, 0x53317B48, 0x3E00DF82,
0x9E5C57BB, 0xCA6F8CA0, 0x1A87562E, 0xDF1769DB,
0xD542A8F6, 0x287EFFC3, 0xAC6732C6, 0x8C4F5573,
0x695B27B0, 0xBBCA58C8, 0xE1FFA35D, 0xB8F011A0,
0x10FA3D98, 0xFD2183B8, 0x4AFCB56C, 0x2DD1D35B,
0x9A53E479, 0xB6F84565, 0xD28E49BC, 0x4BFB9790,
0xE1DDF2DA, 0xA4CB7E33, 0x62FB1341, 0xCEE4C6E8,
0xEF20CADA, 0x36774C01, 0xD07E9EFE, 0x2BF11FB4,
0x95DBDA4D, 0xAE909198, 0xEAAD8E71, 0x6B93D5A0,
0xD08ED1D0, 0xAFC725E0, 0x8E3C5B2F, 0x8E7594B7,
0x8FF6E2FB, 0xF2122B64, 0x8888B812, 0x900DF01C,
0x4FAD5EA0, 0x688FC31C, 0xD1CFF191, 0xB3A8C1AD,
0x2F2F2218, 0xBE0E1777, 0xEA752DFE, 0x8B021FA1,
0xE5A0CC0F, 0xB56F74E8, 0x18ACF3D6, 0xCE89E299,
0xB4A84FE0, 0xFD13E0B7, 0x7CC43B81, 0xD2ADA8D9,
0x165FA266, 0x80957705, 0x93CC7314, 0x211A1477,
        0xE6AD2065, 0x77B5FA86, 0xC75442F5, 0xFB9D35CF,
        0xEBCDAF0C, 0x7B3E89A0, 0xD6411BD3, 0xAE1E7E49,
        0x00250E2D, 0x2071B35E, 0x226800BB, 0x57B8E0AF,
        0x2464369B, 0xF009B91E, 0x5563911D, 0x59DFA6AA,
        0x78C14389, 0xD95A537F, 0x207D5BA2, 0x02E5B9C5,
        0x83260376, 0x6295CFA9, 0x11C81968, 0x4E734A41,
        0xB3472DCA, 0x7B14A94A, 0x1B510052, 0x9A532915,
        0xD60F573F, 0xBC9BC6E4, 0x2B60A476, 0x81E67400,
        0x08BA6FB5, 0x571BE91F, 0xF296EC6B, 0x2A0DD915,
        0xB6636521, 0xE7B9F9B6, 0xFF34052E, 0xC5855664,
        0x53B02D5D, 0xA99F8FA1, 0x08BA4799, 0x6E85076A
    ],
    [
        0x4B7A70E9, 0xB5B32944, 0xDB75092E, 0xC4192623,
        0xAD6EA6B0, 0x49A7DF7D, 0x9CEE60B8, 0x8FEDB266,
        0xECAA8C71, 0x699A17FF, 0x5664526C, 0xC2B19EE1,
        0x193602A5, 0x75094C29, 0xA0591340, 0xE4183A3E,
        0x3F54989A, 0x5B429D65, 0x6B8FE4D6, 0x99F73FD6,
        0xA1D29C07, 0xEFE830F5, 0x4D2D38E6, 0xF0255DC1,
        0x4CDD2086, 0x8470EB26, 0x6382E9C6, 0x021ECC5E,
        0x09686B3F, 0x3EBAEFC9, 0x3C971814, 0x6B6A70A1,
        0x687F3584, 0x52A0E286, 0xB79C5305, 0xAA500737,
        0x3E07841C, 0x7FDEAE5C, 0x8E7D44EC, 0x5716F2B8,
        0xB03ADA37, 0xF0500C0D, 0xF01C1F04, 0x0200B3FF,
        0xAE0CF51A, 0x3CB574B2, 0x25837A58, 0xDC0921BD,
        0xD19113F9, 0x7CA92FF6, 0x94324773, 0x22F54701,
        0x3AE5E581, 0x37C2DADC, 0xC8B57634, 0x9AF3DDA7,
        0xA9446146, 0x0FD0030E, 0xECC8C73E, 0xA4751E41,
        0xE238CD99, 0x3BEA0E2F, 0x3280BBA1, 0x183EB331,
        0x4E548B38, 0x4F6DB908, 0x6F420D03, 0xF60A04BF,
        0x2CB81290, 0x24977C79, 0x5679B072, 0xBCAF89AF,
        0xDE9A771F, 0xD9930810, 0xB38BAE12, 0xDCCF3F2E,
        0x5512721F, 0x2E6B7124, 0x501ADDE6, 0x9F84CD87,
        0x7A584718, 0x7408DA17, 0xBC9F9ABC, 0xE94B7D8C,
        0xEC7AEC3A, 0xDB851DFA, 0x63094366, 0xC464C3D2,
        0xEF1C1847, 0x3215D908, 0xDD433B37, 0x24C2BA16,
        0x12A14D43, 0x2A65C451, 0x50940002, 0x133AE4DD,
        0x71DFF89E, 0x10314E55, 0x81AC77D6, 0x5F11199B,
        0x043556F1, 0xD7A3C76B, 0x3C11183B, 0x5924A509,
        0xF28FE6ED, 0x97F1FBFA, 0x9EBABF2C, 0x1E153C6E,
        0x86E34570, 0xEAE96FB1, 0x860E5E0A, 0x5A3E2AB3,
        0x771FE71C, 0x4E3D06FA, 0x2965DCB9, 0x99E71D0F,
        0x803E89D6, 0x5266C825, 0x2E4CC978, 0x9C10B36A,
        0xC6150EBA, 0x94E2EA78, 0xA5FC3C53, 0x1E0A2DF4,
        0xF2F74EA7, 0x361D2B3D, 0x1939260F, 0x19C27960,
        0x5223A708, 0xF71312B6, 0xEBADFE6E, 0xEAC31F66,
        0xE3BC4595, 0xA67BC883, 0xB17F37D1, 0x018CFF28,
        0xC332DDEF, 0xBE6C5AA5, 0x65582185, 0x68AB9802,
        0xEECEA50F, 0xDB2F953B, 0x2AEF7DAD, 0x5B6E2F84,
        0x1521B628, 0x29076170, 0xECDD4775, 0x619F1510,
        0x13CCA830, 0xEB61BD96, 0x0334FE1E, 0xAA0363CF,
        0xB5735C90, 0x4C70A239, 0xD59E9E0B, 0xCBAADE14,
        0xEECC86BC, 0x60622CA7, 0x9CAB5CAB, 0xB2F3846E,
        0x648B1EAF, 0x19BDF0CA, 0xA02369B9, 0x655ABB50,
        0x40685A32, 0x3C2AB4B3, 0x319EE9D5, 0xC021B8F7,
        0x9B540B19, 0x875FA099, 0x95F7997E, 0x623D7DA8,
        0xF837889A, 0x97E32D77, 0x11ED935F, 0x16681281,
        0x0E358829, 0xC7E61FD6, 0x96DEDFA1, 0x7858BA99,
        0x57F584A5, 0x1B227263, 0x9B83C3FF, 0x1AC24696,
        0xCDB30AEB, 0x532E3054, 0x8FD948E4, 0x6DBC3128,
        0x58EBF2EF, 0x34C6FFEA, 0xFE28ED61, 0xEE7C3C73,
        0x5D4A14D9, 0xE864B7E3, 0x42105D14, 0x203E13E0,
        0x45EEE2B6, 0xA3AAABEA, 0xDB6C4F15, 0xFACB4FD0,
        0xC742F442, 0xEF6ABBB5, 0x654F3B1D, 0x41CD2105,
        0xD81E799E, 0x86854DC7, 0xE44B476A, 0x3D816250,
        0xCF62A1F2, 0x5B8D2646, 0xFC8883A0, 0xC1C7B6A3,
        0x7F1524C3, 0x69CB7492, 0x47848A0B, 0x5692B285,
        0x095BBF00, 0xAD19489D, 0x1462B174, 0x23820E00,
        0x58428D2A, 0x0C55F5EA, 0x1DADF43E, 0x233F7061,
        0x3372F092, 0x8D937E41, 0xD65FECF1, 0x6C223BDB,
        0x7CDE3759, 0xCBEE7460, 0x4085F2A7, 0xCE77326E,
        0xA6078084, 0x19F8509E, 0xE8EFD855, 0x61D99735,
        0xA969A7AA, 0xC50C06C2, 0x5A04ABFC, 0x800BCADC,
        0x9E447A2E, 0xC3453484, 0xFDD56705, 0x0E1E9EC9,
        0xDB73DBD3, 0x105588CD, 0x675FDA79, 0xE3674340,
        0xC5C43465, 0x713E38D8, 0x3D28F89E, 0xF16DFF20,
        0x153E21E7, 0x8FB03D4A, 0xE6E39F2B, 0xDB83ADF7
    ],
    [
        0xE93D5A68, 0x948140F7, 0xF64C261C, 0x94692934,
        0x411520F7, 0x7602D4F7, 0xBCF46B2E, 0xD4A20068,
        0xD4082471, 0x3320F46A, 0x43B7D4B7, 0x500061AF,
        0x1E39F62E, 0x97244546, 0x14214F74, 0xBF8B8840,
        0x4D95FC1D, 0x96B591AF, 0x70F4DDD3, 0x66A02F45,
        0xBFBC09EC, 0x03BD9785, 0x7FAC6DD0, 0x31CB8504,
        0x96EB27B3, 0x55FD3941, 0xDA2547E6, 0xABCA0A9A,
        0x28507825, 0x530429F4, 0x0A2C86DA, 0xE9B66DFB,
        0x68DC1462, 0xD7486900, 0x680EC0A4, 0x27A18DEE,
0x4F3FFEA2, 0xE887AD8C, 0xB58CE006, 0x7AF4D6B6,
0xAACE1E7C, 0xD3375FEC, 0xCE78A399, 0x406B2A42,
0x20FE9E35, 0xD9F385B9, 0xEE39D7AB, 0x3B124E8B,
0x1DC9FAF7, 0x4B6D1856, 0x26A36631, 0xEAE397B2,
0x3A6EFA74, 0xDD5B4332, 0x6841E7F7, 0xCA7820FB,
0xFB0AF54E, 0xD8FEB397, 0x454056AC, 0xBA489527,
0x55533A3A, 0x20838D87, 0xFE6BA9B7, 0xD096954B,
0x55A867BC, 0xA1159A58, 0xCCA92963, 0x99E1DB33,
0xA62A4A56, 0x3F3125F9, 0x5EF47E1C, 0x9029317C,
0xFDF8E802, 0x04272F70, 0x80BB155C, 0x05282CE3,
0x95C11548, 0xE4C66D22, 0x48C1133F, 0xC70F86DC,
0x07F9C9EE, 0x41041F0F, 0x404779A4, 0x5D886E17,
0x325F51EB, 0xD59BC0D1, 0xF2BCC18F, 0x41113564,
0x257B7834, 0x602A9C60, 0xDFF8E8A3, 0x1F636C1B,
0x0E12B4C2, 0x02E1329E, 0xAF664FD1, 0xCAD18115,
0x6B2395E0, 0x333E92E1, 0x3B240B62, 0xEEBEB922,
0x85B2A20E, 0xE6BA0D99, 0xDE720C8C, 0x2DA2F728,
0xD0127845, 0x95B794FD, 0x647D0862, 0xE7CCF5F0,
0x5449A36F, 0x877D48FA, 0xC39DFD27, 0xF33E8D1E,
0x0A476341, 0x992EFF74, 0x3A6F6EAB, 0xF4F8FD37,
0xA812DC60, 0xA1EBDDF8, 0x991BE14C, 0xDB6E6B0D,
0xC67B5510, 0x6D672C37, 0x2765D43B, 0xDCD0E804,
0xF1290DC7, 0xCC00FFA3, 0xB5390F92, 0x690FED0B,
0x667B9FFB, 0xCEDB7D9C, 0xA091CF0B, 0xD9155EA3,
0xBB132F88, 0x515BAD24, 0x7B9479BF, 0x763BD6EB,
0x37392EB3, 0xCC115979, 0x8026E297, 0xF42E312D,
0x6842ADA7, 0xC66A2B3B, 0x12754CCC, 0x782EF11C,
0x6A124237, 0xB79251E7, 0x06A1BBE6, 0x4BFB6350,
0x1A6B1018, 0x11CAEDFA, 0x3D25BDD8, 0xE2E1C3C9,
0x44421659, 0x0A121386, 0xD90CEC6E, 0xD5ABEA2A,
0x64AF674E, 0xDA86A85F, 0xBEBFE988, 0x64E4C3FE,
0x9DBC8057, 0xF0F7C086, 0x60787BF8, 0x6003604D,
0xD1FD8346, 0xF6381FB0, 0x7745AE04, 0xD736FCCC,
0x83426B33, 0xF01EAB71, 0xB0804187, 0x3C005E5F,
0x77A057BE, 0xBDE8AE24, 0x55464299, 0xBF582E61,
0x4E58F48F, 0xF2DDFDA2, 0xF474EF38, 0x8789BDC2,
0x5366F9C3, 0xC8B38E74, 0xB475F255, 0x46FCD9B9,
0x7AEB2661, 0x8B1DDF84, 0x846A0E79, 0x915F95E2,
0x466E598E, 0x20B45770, 0x8CD55591, 0xC902DE4C,
0xB90BACE1, 0xBB8205D0, 0x11A86248, 0x7574A99E,
0xB77F19B6, 0xE0A9DC09, 0x662D09A1, 0xC4324633,
0xE85A1F02, 0x09F0BE8C, 0x4A99A025, 0x1D6EFE10,
0x1AB93D1D, 0x0BA5A4DF, 0xA186F20F, 0x2868F169,
0xDCB7DA83, 0x573906FE, 0xA1E2CE9B, 0x4FCD7F52,
        0x50115E01, 0xA70683FA, 0xA002B5C4, 0x0DE6D027,
        0x9AF88C27, 0x773F8641, 0xC3604C06, 0x61A806B5,
        0xF0177A28, 0xC0F586E0, 0x006058AA, 0x30DC7D62,
        0x11E69ED7, 0x2338EA63, 0x53C2DD94, 0xC2C21634,
        0xBBCBEE56, 0x90BCB6DE, 0xEBFC7DA1, 0xCE591D76,
        0x6F05E409, 0x4B7C0188, 0x39720A3D, 0x7C927C24,
        0x86E3725F, 0x724D9DB9, 0x1AC15BB4, 0xD39EB8FC,
        0xED545578, 0x08FCA5B5, 0xD83D7CD3, 0x4DAD0FC4,
        0x1E50EF5E, 0xB161E6F8, 0xA28514D9, 0x6C51133C,
        0x6FD5C7E7, 0x56E14EC4, 0x362ABFCE, 0xDDC6C837,
        0xD79A3234, 0x92638212, 0x670EFA8E, 0x406000E0
    ],
    [
        0x3A39CE37, 0xD3FAF5CF, 0xABC27737, 0x5AC52D1B,
        0x5CB0679E, 0x4FA33742, 0xD3822740, 0x99BC9BBE,
        0xD5118E9D, 0xBF0F7315, 0xD62D1C7E, 0xC700C47B,
        0xB78C1B6B, 0x21A19045, 0xB26EB1BE, 0x6A366EB4,
        0x5748AB2F, 0xBC946E79, 0xC6A376D2, 0x6549C2C8,
        0x530FF8EE, 0x468DDE7D, 0xD5730A1D, 0x4CD04DC6,
        0x2939BBDB, 0xA9BA4650, 0xAC9526E8, 0xBE5EE304,
        0xA1FAD5F0, 0x6A2D519A, 0x63EF8CE2, 0x9A86EE22,
        0xC089C2B8, 0x43242EF6, 0xA51E03AA, 0x9CF2D0A4,
        0x83C061BA, 0x9BE96A4D, 0x8FE51550, 0xBA645BD6,
        0x2826A2F9, 0xA73A3AE1, 0x4BA99586, 0xEF5562E9,
        0xC72FEFD3, 0xF752F7DA, 0x3F046F69, 0x77FA0A59,
        0x80E4A915, 0x87B08601, 0x9B09E6AD, 0x3B3EE593,
        0xE990FD5A, 0x9E34D797, 0x2CF0B7D9, 0x022B8B51,
        0x96D5AC3A, 0x017DA67D, 0xD1CF3ED6, 0x7C7D2D28,
        0x1F9F25CF, 0xADF2B89B, 0x5AD6B472, 0x5A88F54C,
        0xE029AC71, 0xE019A5E6, 0x47B0ACFD, 0xED93FA9B,
        0xE8D3C48D, 0x283B57CC, 0xF8D56629, 0x79132E28,
        0x785F0191, 0xED756055, 0xF7960E44, 0xE3D35E8C,
        0x15056DD4, 0x88F46DBA, 0x03A16125, 0x0564F0BD,
        0xC3EB9E15, 0x3C9057A2, 0x97271AEC, 0xA93A072A,
        0x1B3F6D9B, 0x1E6321F5, 0xF59C66FB, 0x26DCF319,
        0x7533D928, 0xB155FDF5, 0x03563482, 0x8ABA3CBB,
        0x28517711, 0xC20AD9F8, 0xABCC5167, 0xCCAD925F,
        0x4DE81751, 0x3830DC8E, 0x379D5862, 0x9320F991,
        0xEA7A90C2, 0xFB3E7BCE, 0x5121CE64, 0x774FBE32,
        0xA8B6E37E, 0xC3293D46, 0x48DE5369, 0x6413E680,
        0xA2AE0810, 0xDD6DB224, 0x69852DFD, 0x09072166,
        0xB39A460A, 0x6445C0DD, 0x586CDECF, 0x1C20C8AE,
        0x5BBEF7DD, 0x1B588D40, 0xCCD2017F, 0x6BB4E3BB,
        0xDDA26A7E, 0x3A59FF45, 0x3E350A44, 0xBCB4CDD5,
        0x72EACEA8, 0xFA6484BB, 0x8D6612AE, 0xBF3C6F47,
        0xD29BE463, 0x542F5D9E, 0xAEC2771B, 0xF64E6370,
        0x740E0D8D, 0xE75B1357, 0xF8721671, 0xAF537D5D,
        0x4040CB08, 0x4EB4E2CC, 0x34D2466A, 0x0115AF84,
        0xE1B00428, 0x95983A1D, 0x06B89FB4, 0xCE6EA048,
        0x6F3F3B82, 0x3520AB82, 0x011A1D4B, 0x277227F8,
        0x611560B1, 0xE7933FDC, 0xBB3A792B, 0x344525BD,
        0xA08839E1, 0x51CE794B, 0x2F32C9B7, 0xA01FBAC9,
        0xE01CC87E, 0xBCC7D1F6, 0xCF0111C3, 0xA1E8AAC7,
        0x1A908749, 0xD44FBD9A, 0xD0DADECB, 0xD50ADA38,
        0x0339C32A, 0xC6913667, 0x8DF9317C, 0xE0B12B4F,
        0xF79E59B7, 0x43F5BB3A, 0xF2D519FF, 0x27D9459C,
        0xBF97222C, 0x15E6FC2A, 0x0F91FC71, 0x9B941525,
        0xFAE59361, 0xCEB69CEB, 0xC2A86459, 0x12BAA8D1,
        0xB6C1075E, 0xE3056A0C, 0x10D25065, 0xCB03A442,
        0xE0EC6E0E, 0x1698DB3B, 0x4C98A0BE, 0x3278E964,
        0x9F1F9532, 0xE0D392DF, 0xD3A0342B, 0x8971F21E,
        0x1B0A7441, 0x4BA3348C, 0xC5BE7120, 0xC37632D8,
        0xDF359F8D, 0x9B992F2E, 0xE60B6F47, 0x0FE3F11D,
        0xE54CDA54, 0x1EDAD891, 0xCE6279CF, 0xCD3E7E6F,
        0x1618B166, 0xFD2C1D05, 0x848FD2C5, 0xF6FB2299,
        0xF523F357, 0xA6327623, 0x93A83531, 0x56CCCD02,
        0xACF08162, 0x5A75EBB5, 0x6E163697, 0x88D273CC,
        0xDE966292, 0x81B949D0, 0x4C50901B, 0x71C65614,
        0xE6C6C7BD, 0x327A140A, 0x45E1D006, 0xC3F27B9A,
        0xC9AA53FD, 0x62A80F00, 0xBB25BFE2, 0x35BDD2F6,
        0x71126905, 0xB2040222, 0xB6CBCF7C, 0xCD769C2B,
        0x53113EC0, 0x1640E3D3, 0x38ABBD60, 0x2547ADF0,
        0xBA38209C, 0xF746CE76, 0x77AFA1C5, 0x20756060,
        0x85CBFE4E, 0x8AE88DD8, 0x7AAAF9B0, 0x4CF9AA7E,
        0x1948C25C, 0x02FB8A8C, 0x01C36AE4, 0xD6EBE1F9,
        0x90D4F869, 0xA65CDEA0, 0x3F09252D, 0xC208E69F,
        0xB74E6132, 0xCE77E25B, 0x578FDFE3, 0x3AC372E6
    ]
]


key = [ 0x4B7A70E9, 0xB5B32944, 0xDB75092E, 0xC4192623,
        0xAD6EA6B0, 0x49A7DF7D, 0x9CEE60B8, 0x8FEDB266,
        0xECAA8C71, 0x699A17FF, 0x5664526C, 0xC2B19EE1,
        0x193602A5, 0x75094C29]

def calculate(L):
    temp = s[0][L >> 24]
    temp = (temp + s[1][L >> 16 & 0xff]) % (0x1<<32)
    temp = temp ^ s[2][L >> 8 & 0xff]
    temp = (temp + s[3][L & 0xff]) % (0x1<<32)
    return temp

def encrypt(data):
        L = data>>32
        R = data & 0xffffffff
        for i in range(0,16):
                L = L^p[i]
                L1 = calculate(L)
                R = R^calculate(L1)
                L,R = R,L
        L,R = R,L
        L = L^p[17]
        R = R^p[16]
        encrypted = (L<<32) ^ R
        return encrypted

def decrypt(data):
    L = data >> 32
    R = data & 0xffffffff
    for i in range(17, 1, -1):
        L = p[i]^L
        L1 = calculate(L)
        R = R^calculate(L1)
        L,R = R,L

    L,R = R,L
    L = L^p[0]
    R = R^p[1]
    data_decrypted1 = (L<<32) ^ R
    return data_decrypted1

# Main
for i in range(0,18):
    p[i] = p[i]^key[i%14]
x = 0
data = 0
for i in range(0,9):
    temp = encrypt(data)
    p[x] = temp >> 32
    x+=1
    p[x] = temp & 0xffffffff
    x+=1
    data = temp
encrypt_data = int(input("Enter data to encrypt: "))
if encrypt_data.bit_length() <=63:
    print("Valid Input!!!")
else:
    print("Invalid Input!!")

data_encrypted = encrypt(encrypt_data)
print("Encrypted data is: ",data_encrypted)
print("Hex value :",hex(data_encrypted))
data_decrypted = decrypt(data_encrypted)
print("Data after decryption is : ",data_decrypted)
```



```
Enter data to encrypt: 1111
Valid Input!!!
Encrypted data is:  17315834592016220924
Hex value : 0xf04e33d1768c42fc
Data after decryption is :  1111
>
```