**CN LAB MANUAL**

1. a) Implementation of the Data Link Layer Framing methods Bit stuffing.

**Aim : Program to Implementation of the Data Link Layer Framing methods Bit stuffing**

**Program : Code**

#include<stdio.h>

#include<string.h>

void main()

{

int a[20],b[30],i,j,k,count,n;

printf("Enter frame length:");

scanf("%d",&n);

printf("Enter input frame (0's & 1's only):");

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

scanf("%d",&a[i]);

i=0; count=1; j=0;

while(i<n)

{

if(a[i]==1)

{

b[j]=a[i];

for(k=i+1;a[k]==1 && k<n && count<5;k++)

{

j++;

b[j]=a[k];

count++;

if(count==5)

{

j++;

b[j]=0;

}

i=k;

}}

else

{

b[j]=a[i];

}

i++;

j++;

}

printf("After stuffing the frame is:");

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

printf("%d",b[i]);

}

Out Put:

Enter frame length:6

Enter input frame (0's & 1's only):1

1

1

1

1

1

After stuffing the frame is:1111101

**1.b) Implementation of the Data Link Layer Framing methods Character Stuffing .**

**Aim : Program to Implementation of the Data Link Layer Framing methods Character Stuffing**

**Program : Code**

#include<stdio.h>

#include<string.h>

#include<process.h>

void main()

{

int i=0,j=0,n,pos;

char a[20],b[50],ch;

printf("Enter string\n");

scanf("%s",&a);

n=strlen(a);

printf("Enter position\n");

scanf("%d",&pos);

if(pos>n)

{

printf("invalid position, Enter again :");

scanf("%d",&pos);}

printf("Enter the character\n");

ch=getche();

b[0]='d';

b[1]='l';

b[2]='e';

b[3]='s';

b[4]='t';

b[5]='x';

j=6;

while(i<n)

{

if(i==pos-1)

{

b[j]='d';

b[j+1]='l';

b[j+2]='e';

b[j+3]=ch;

b[j+4]='d';

b[j+5]='l';

b[j+6]='e';

j=j+7;

}

if(a[i]=='d' && a[i+1]=='l' && a[i+2]=='e')

{

b[j]='d';

b[j+1]='l';

b[j+2]='e';

j=j+3;

}

b[j]=a[i];

i++;

j++;

}

b[j]='d';

b[j+1]='l';

b[j+2]='e';

b[j+3]='e';

b[j+4]='t';

b[j+5]='x';

b[j+6]='\0';

printf("\nframe after stuffing:\n");

printf("%s",b);

}

Enter string

SVCETA

Enter position

2

Enter the character

frame after stuffing:

dlestxSdldleVCETAdleetx

**//2. CRC PROGRAM (Transmitter and Receiver)**

2. Implementation of CRC polynomials, CRC 12, CRC 16 and CRC CCIP.

Aim: program to Implementation of CRC polynomials, CRC 12, CRC 16 and CRC CCIP.

**#include<stdio.h>**

**#include<stdlib.h>**

**int print(int \*a,int n)**

**{**

**int i;**

**for(i=0;i<n;i++)**

**{**

**printf("%d",a[i]);**

**}**

**printf("\n");**

**return (0);**

**}**

**int crc(int \*g,int \*q,int \*r,int ng,int nq)**

**{**

**int i,j,k;**

**k = 0;**

**for(i=0;i<nq;i++)**

**{**

**if(r[i]==0)**

**{**

**for(j=i;j<(i+ng);j++)**

**{**

**r[j]=r[j]^0;**

**}**

**q[i]=0;**

**}**

**else**

**{**

**for(j=i;j<(i+ng);j++)**

**{**

**r[j]=r[j]^g[k];**

**k++;**

**}**

**q[i]=1;**

**k=0;**

**}**

**}**

**return (0);**

**}**

**main()**

**{**

**int \*gx,\*tx,\*q,\*r;**

**int i,j,nt,ng,nq,n,flag=1;**

**printf("\n Enter no. of bits in message to be transmitted:");**

**scanf("%d",&nt);**

**printf("\n Enter no. of bits in G(x) :");**

**scanf("%d",&ng);**

**n=nt+ng-1;**

**nq=nt;**

**gx=malloc(sizeof(int)\*ng);**

**tx=malloc(sizeof(int)\*n);**

**r=malloc(sizeof(int)\*n);**

**q=calloc(nq,sizeof(int));**

**printf("\n Enter message :");**

**for(i=0;i<nt;i++)**

**{**

**scanf("%d",&tx[i]);**

**r[i]=tx[i];**

**}**

**for(;i<n;i++)**

**{**

**r[i]=0;**

**}**

**printf("\n Enter G(x) :");**

**for(i=0;i<ng;i++)**

**{**

**scanf("%d",&gx[i]);**

**}**

**/\* \*\*\*\*\*AT TRANSMITTER\*\*\*\*\* \*/**

**printf("\n CRC at transmitter :");**

**printf("\n Message to be transmitted :");**

**print(tx,nt);**

**printf("\n G(x)=");**

**print(gx,ng);**

**printf("\n Message with '0' appended :");**

**print(r,n);**

**crc(gx,q,r,ng,nq);**

**printf("\n Quotient at transmitter:");**

**print(q,nq);**

**printf("\n Remainder at transmitter :");**

**print(r,n);**

**for(i=0;i<nt;i++)**

**{**

**r[i]=tx[i];**

**}**

**printf("\n Transmitted message :");**

**print(r,n);**

**/\* \*\*\*\*\* AT RECEIVER\*\*\*\*\* \*/**

**printf("\n CRC at receiver :");**

**printf("\n Message received :");**

**print(r,n);**

**printf("\n G(x)=");**

**print(gx,ng);**

**crc(gx,q,r,ng,nq);**

**printf("\n Quotient at receiver :");**

**print(q,nq);**

**printf("\n Remainder at receiver :");**

**print(r,n);**

**for(i=0;i<n;i++)**

**{**

**if(r[i]!=0)**

**{**

**flag=0;**

**break;**

**}**

**}**

**if(flag)**

**printf("\n No error detected -->CRC algorithm implemented successfully.");**

**else**

**printf("\n Error detected.");**

**getch();**

**}**

**Sample Input and Output**

Enter no. of bits in message to be transmitted:8

Enter no. of bits in G(x) :5

Enter message :0 1 1 0 1 1 0 1

Enter G(x) :1 0 1 0 1

CRC at transmitter :

Message to be transmitted :01101101

G(x)=10101

Message with '0' appended :011011010000

Quotient at transmitter:01110111

Remainder at transmitter :000000001011

Transmitted message :011011011011

CRC at receiver :

Message received :011011011011

G(x)=10101

Quotient at receiver :01110111

Remainder at receiver :000000000000

No error detected -->CRC algorithm implemented successfully.

3. Implementation of Sliding Window Protocol Select Repeat ARQ

Aim : Implementation of Sliding Window Protocol Select Repeat ARQ

Program :

#include<stdio.h>

int main()

{

int w,i,f,frames[50];

printf("Enter window size: ");

scanf("%d",&w);

printf("\nEnter number of frames to transmit: ");

scanf("%d",&f);

printf("\nEnter %d frames: ",f);

for(i=1;i<=f;i++)

scanf("%d",&frames[i]);

printf("\nWith sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)\n\n");

printf("After sending %d frames at each stage sender waits for acknowledgement sent by the receiver\n\n",w);

for(i=1;i<=f;i++)

{

if(i%w==0)

{

printf("%d\n",frames[i]);

printf("Acknowledgement of above frames sent is received by sender\n\n");

}

else

printf("%d ",frames[i]);

}

if(f%w!=0)

printf("\nAcknowledgement of above frames sent is received by sender\n");

return 0;

}

Output:

Enter window size: 3

Enter number of frames to transmit: 5

Enter 5 frames: 5 6 7 8 9

With sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)

After sending 3 frames at each stage sender waits for acknowledgement sent by the receiver

5 6 7  
Acknowledgement of above frames sent is received by sender

8 9  
Acknowledgement of above frames sent is received by sender

**AIM**

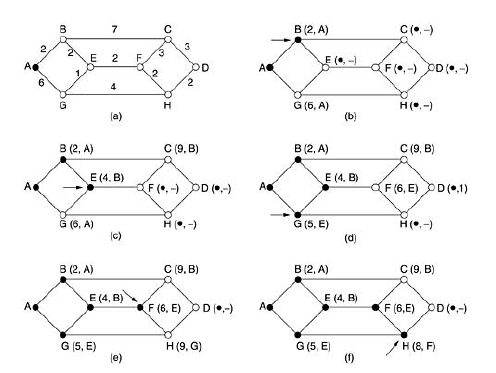
**4.Implement Dijkstra ‘s algorithm to compute the Shortest path thru a graph.**

**Description**

The idea is to build a graph of thesubnet, with each node of the graph representing a router and each arc of the graph representing a communication line (often called a link). To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.

To implement Dikstra algorithms for computing the shortest path between two nodes of a graph are known.

* Each node is labeled (in parentheses) with its distance from the source node along the best known path.
* Initially, no paths are known, so all nodes are labeled with infinity
* Algorithm proceeds and paths are found, the labels may change, reflecting better paths.
* Initially, all labels are tentative. When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

****

* Find the shortest path from *A* to *D*.
* Marking as node *A* as permanent, indicated by a filled-in circle.
* Examine, each of the nodes adjacent to *A* (the working node), relabeling each one
* with the distance to *A*.
* Examine all the tentatively labeled nodes in the whole graph and make the one with the smallest label permanent. This one becomes the new working node.
* After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively-labeled node with the small value

**Program:**

#include<conio.h>

#include<stdio.h>

static int dsp[10][10],nodes,perm,tem;

struct

{

char src;

char dest;

int length;

} stemp,permanent[10]={' ',' ',0},temp[10]={' ',' ',-1};

void sort()

{

int i,j,k;

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

{

k=1;

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

{

if((temp[j].length<=temp[j+1].length))

{

stemp=temp[j];

temp[j]=temp[j+1];

temp[j+1]=stemp;

k=0;

}

}

if(k)

break;

}

permanent[perm++]=temp[tem-1];

temp[tem-1].src=' ';

temp[tem-1].dest=' ';

temp[tem-1].length=-1;

tem--;

}

void main()

{

int i,j,k,l,m,n=0,point;

char initial,dest,path[10]={' '};

clrscr();

printf("\t\t Shortest path dijkstra's algothrim");

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Enter the number of nodes:\n");

scanf("%d",&nodes);

printf("\nEnter the adjacency matrix");

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

{

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

scanf("%d",&dsp[i][j]);

}

fflush(stdin);

printf("\n Enter the source node");

scanf("%c",&initial);

fflush(stdin);

printf("\n Enter the destination node\n");

scanf("%c",&dest);

permanent[perm].src=initial;

permanent[perm].dest=initial;

permanent[perm++].length=0;

i=permanent[perm-1].dest-97;

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

{

if(i!=j)

{

if(dsp[i][j]>0)

{

temp[tem].src=permanent[perm-1].src;

temp[tem].dest=j+97;

temp[tem++].length=dsp[i][j];

}

}

}

sort();

while(tem>=0)

{

j=permanent[perm-1].dest-97;

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

{

if(i!=initial-97)

{

if(dsp[j][i]>0)

{

l=-1;

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

{

if(permanent[k].dest==(i+97))

l=k;

}

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

{

if(temp[k].dest==(i+97))

l=k;

}

if(l<0)

{

temp[tem].src=j+97;

temp[tem].dest=i+97;

for(m=0;m<perm;m++)

{

if(permanent[m].dest==temp[tem].src)

n=permanent[m].length;

}

temp[tem++].length=dsp[j][i]+n;

}

else

{

for(m=0;m<perm;m++)

{

if(permanent[m].dest==j+97)

{

n=permanent[m].length+dsp[j][i];

break;

}

else

n=dsp[j][i];

}

if((n<temp[l].length))

{

temp[l].length=n;

temp[l].src=j+97;

tpaemp[l].dest=i+97;

} } } } }

sort();

}

printf("\n shortest path :\n\n");

printf("from %c to %c is:",initial,dest);

for(i=0;i<perm-1;i++)

{

if(permanent[i].dest==dest)

{

point=i;

n=i;

break;

}

}

i=0;

for(j=perm;j>0;j--)

{

if(permanent[j-1].dest==permanent[point].src)

{

path[i++]=permanent[point].dest;

point=j-1;

} }

path[i]=initial;

for(j=i;j>=0;j--)

{

printf("%c",path[j]);

if(j>0)

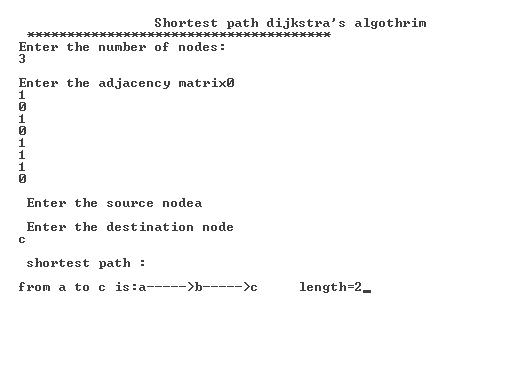
printf("----->"); }

printf("\t length=%d",permanent[m].length);

getch();

}

Output:



5. Implementation Link State Routing algorithm.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*  C Program to implement link state routing.\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

main()

{

    int n,a[10][10],i,j,k;

    printf("\n ENTER THE NO.OF NODES: ");

    scanf("%d",&n);

    printf("\n ENTER THE MATRIX ELEMENTS: ");

    for(i=0;i<n;i++)

    {

        printf("\nENTER THE DISTANCE FOR NODE:%d\n",i+1);

        for(j=0;j<n;j++)

        {

            scanf("%d",&a[i][j]);

        }

    }

    for(i=0;i<n;i++)

    {

        printf("THE LINK STATE STATE PACKETS FOR NODE:%d\n",i+1);

        printf("\n NODE\tDISTANCE\n");

        for(j=0;j<n;j++)

        {

            if(a[i][j]!=0&&a[i][j]!=-1)

            {

                printf("%d\t%d\n",j+1,a[i][j]);

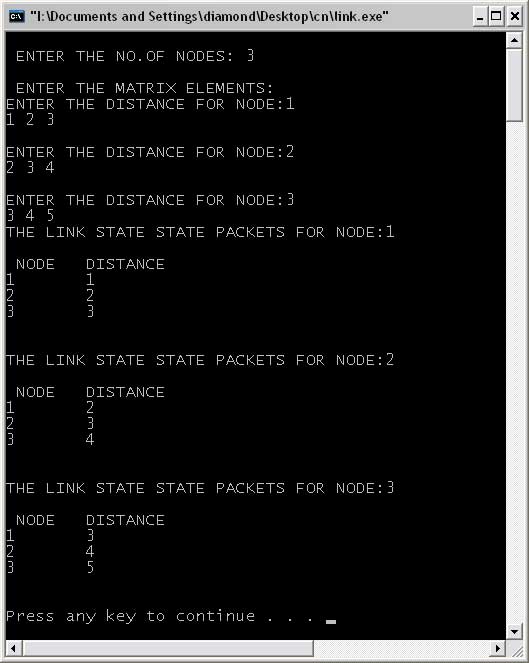
            }

        }

        printf("\n\n");

    }

}



6.Program to obtain Routing table for each node using the Distance Vector Routing algorithm of a given subnet.

**AIM**

**Obtaining the vector routing table using Distance Vector Routing Algorithm by taking an example subnet graph with weights .**

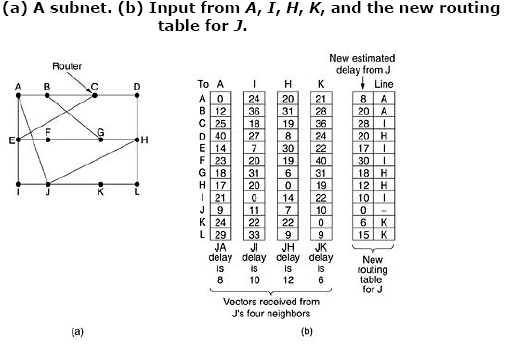
**Description**

**Distance vector routing** algorithms operate by having each router maintain a table (i.e, a vector) giving the best known distance to each destination and which line to use to get there. These tables are updated by exchanging information with the neighbors.In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in the subnet. This entry contains two parts: the preferred outgoing line to use for that destination and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

By performing this calculation for each neighbor,a router can find out which estimate seems the best and use that estimate and the corresponding line in its new routing table. Note that the old routing table is not used in the calculation.

The first four columns of part (b) show the delay vectors received from the neighbors of router *J*. *A* claims to have a 12-msec delay to *B*, a 25-msec delay to *C*, a 40-msec delay to *D*, etc. Suppose that *J* has measured or estimated its delay to its neighbors, *A*, *I, H*, and *K* as 8, 10, 12, and 6 msec, respectively.

Consider how *J* computes its new route to router *G*. It knows that it can get to *A* in 8 msec, and*A* claims to be able to get to *G* in 18 msec, so *J* knows it can count on a delay of 26 msec to *G* if it forwards packets bound for *G* to *A*. Similarly, it computes the delay to *G* via *I*, *H*, and *K* as 41 (31 + 10), 18 (6 + 12), and 37 (31 + 6) msec, respectively. The best of these values is 18, so it makes an entry in its routing table that the delay to *G* is 18 msec and that the route to use is via *H*. The same calculation is performed for all the other destinations.

****

**Program:**

#include"stdio.h"

#include"conio.h"

void main()

{

int i=0,j=0,nei,n,ah[20];

int v[50][50],r[20],min=10000,si;

char s[50],dest,line[50];

for(i=0;i<50;i++)

s[i]= NULL;

clrscr();

printf("\nEnter the no of nodes:\n");

scanf("%d",&n);

printf("\nEnter the destination:\n");

scanf("%s",&dest);

printf("\nEnter the no.of neighbours:\n");

scanf("%d",&nei);

printf("\nEnter the neighbours & slash(0) at end:\n");

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

scanf("%s",s[i]);

for(j=0;j<nei;)

{

printf("\nThe distance from %c to %s :",dest,s[j]);

scanf("%d",&ah[j]);

j++; }

printf("\nEnter the table of entries:");

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

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

scanf("%d",&v[i][j]);

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

{

min=10000;

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

if(v[i][j]+ah[j]<min)

{

min=v[i][j]+ah[j];

line[i]=s[j];

}r[i]=min;

if(dest-97==i)

{ r[i]=0;

line[i]=' ';

}

}

printf("\n new estimated delays from %c |",dest);

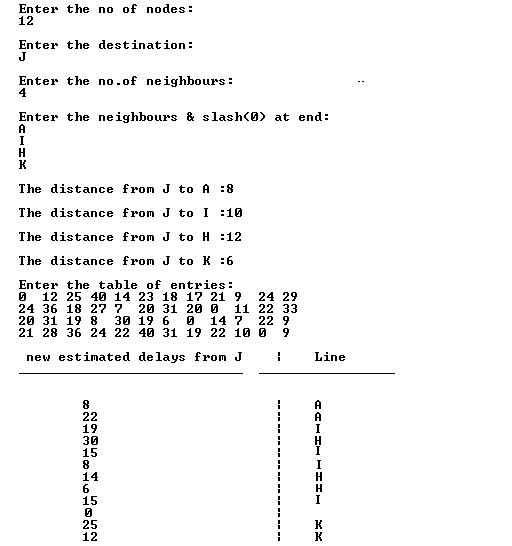
printf(" Line\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

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

printf("\n \t%d\t\t\t| %s",r[i],line[i]);

getch();

}

**OUTPUT:**

7.Implementation of encryption & decryption using DES algorithm.

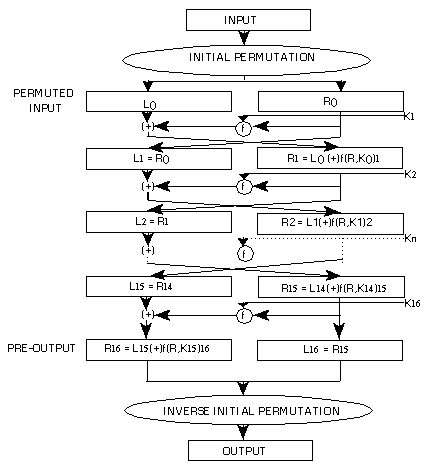
Program 7.Implementation of encryption & decryption using DES algorithm.

**AIM**

**Take a 64 bit playing text and encrypt the same using DES algorithm**

**Description:**

The **Data Encryption Standard algorithm** is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64-bit key. A key consists of 64 binary digits ("O"s or "1"s) of which 56 bits are randomly generated and used directly by the algorithm. The other 8 bits, which are not used by the algorithm, are used for error detection. The 8 error detecting bits are set to make the parity of each 8-bit byte of the key odd, i.e., there is an odd number of "1"s in each 8-bit byte1. Deciphering must be accomplished by using the same key as for enciphering, but with the schedule of addressing the key bits altered so that the deciphering process is the reverse of the enciphering process. A block to be enciphered is subjected to an initial permutation **IP**, then to a complex key-dependent computation and finally to a permutation which is the inverse of the initial permutation **IP-1**. The key-dependent computation can be simply defined in terms of a function ***f***, called the cipher function, and a function **KS**, called the key schedule. A description of the computation is given first, along with details as to how the algorithm is used for encipherment. Next, the use of the algorithm for decipherment is described. Finally, a definition of the cipher function ***f*** is given in terms of primitive functions which are called the selection functions **Si** and the permutation function **P. Si, P** and **KS** of the algorithm are contained in the Appendix.

The following notation is convenient: Given two blocks **L** and **R** of bits, **LR** denotes the block consisting of the bits of **L** followed by the bits of **R**. Since concatenation is associative, B1B2...B8, for example, denotes the block consisting of the bits of B1 followed by the bits of B2...followed by the bits of B8.   


**Figure 1. Enciphering computation.**

.

**Program**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

void main()

{

int i,ch,lp;

char cipher[50],plain[50];

char key[50];

clrscr();

while(1)

{

printf("\n-----MENU-----");

printf("\n1:Data Encryption\t2:Data Decryption\t3:Exit");

printf("\nEnter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nData Encryption");

printf("\nEnter the plain text:");

fflush(stdin);

gets(plain);

printf("\nEnter the encryption key:");

gets(key);

lp=strlen(key);

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

cipher[i]=plain[i]^lp;

cipher[i]='\0';

printf("\nThe encrypted text is:");

puts(cipher);

break;

case 2: printf("\nData decryption");

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

plain[i]=cipher[i]^lp;

printf("\nDecrypted text is:");

puts(plain);

break;

case 3: exit(0);

}

}

}

**Output:**

**-------Menu--------**

1. **Data Encryption**
2. **Data Decryption**
3. **Exit**

**Enter your Choice: 1**

**Data Encryption**

**Enter the plain text: rupesh**

**Enter the encryption key: hsepur**

**The encrypted text is: tsvcun**

**-------Menu--------**

**1. Data Encryption**

**2. Data Decryption**

**3. Exit**

**Enter your Choice: 2**

**Data Decryption**

**Decrypted text is: rupesh**

**-------Menu--------**

**1. Data Encryption**

**2. Data Decryption**

**3. Exit**

**Enter your Choice: 3**

**Program 8. Implementation of encryption & decryption mechanisms using RSA algorithm.**

**AIM**

**Using RSA algorithm Encrypt a text data and Decrypt the same**

Program:

#include< stdio.h>  
#include< conio.h>  
  
int phi,M,n,e,d,C,FLAG;  
  
void check()  
{  
int i;  
for(i=3;e%i==0 && phi%i==0;i+2)  
{  
FLAG = 1;  
return;  
}  
FLAG = 0;  
}  
  
void encrypt()  
{  
int i;  
C = 1;  
for(i=0;i< e;i++)  
C=C\*M%n;  
C = C%n;  
printf("\n\tEncrypted [keyword](http://cppgm.blogspot.com/2008/01/rsa-algorithm.html) : %d",C);  
}  
  
void decrypt()  
{  
int i;  
M = 1;  
for(i=0;i< d;i++)  
M=M\*C%n;  
M = M%n;  
printf("\n\tDecrypted keyword : %d",M);  
}  
  
void main()  
{  
int p,q,s;  
clrscr();  
printf("Enter Two Relatively Prime Numbers\t: ");  
scanf("%d%d",&p,&q);  
n = p\*q;  
phi=(p-1)\*(q-1);  
printf("\n\tF(n)\t= %d",phi);  
do  
{  
printf("\n\nEnter e\t: ");  
scanf("%d",&e);  
check();  
}while(FLAG==1);  
d = 1;  
do  
{  
s = (d\*e)%phi;  
d++;  
}while(s!=1);  
d = d-1;  
printf("\n\tPublic Key\t: {%d,%d}",e,n);  
printf("\n\tPrivate Key\t: {%d,%d}",d,n);  
printf("\n\nEnter The Plain Text\t: ");  
scanf("%d",&M);  
encrypt();  
printf("\n\nEnter the Cipher text\t: ");  
scanf("%d",&C);  
decrypt();  
getch();  
}  
  
 **OUTPUT** :

**Enter Two Relatively Prime Numbers : 7 17  
  
F(n) = 96   
  
Enter e : 5   
  
Public Key : {5,119}  
Private Key : {77,119}  
  
Enter The Plain Text : 19  
  
Encrypted keyword : 66  
  
Enter the Cipher text : 66**