**COMPUTER NETWORKS LAB MANUAL**

**Exp No : 1(a)**

**Aim :**

Write a program to implement the data link layer framing method such as bit stuffing.

**Description :**

new technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary no of bits per character. Each frame begins and ends with the special bit pattern, 01111110, called a flag byte. Whenever the sender's data link layer encounters five consecutive ones in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This bit stuffing is analogous to character stuffing, in which a DLE is stuffed into the outgoing character stream before DLE in the data.

The

**Program :**

#include<stdio.h>

void main()

{

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

printf("Enter the frame size : ");

scanf("%d",&n);

printf("Enter the frame in the form of 0's and 1's : ");

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 bit stuffing : ");

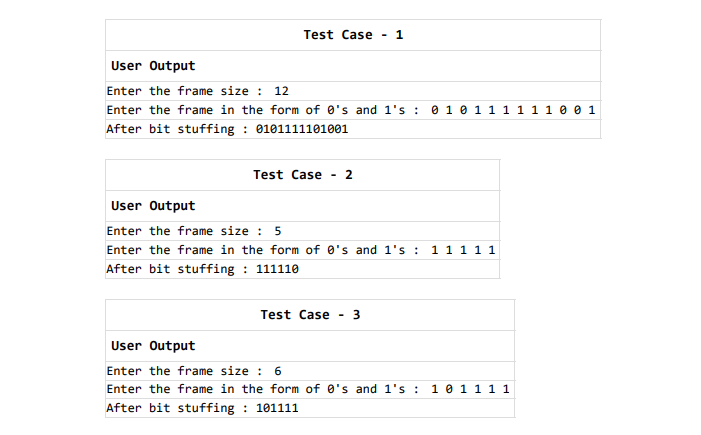
for(i=0;i<j;i++)

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

printf("\n");

}

**Output :**

****

**Exp No : 1(b)**

**Aim :**

Write a program to implement the data link layer framing method such as character stuffing and also de stuff it.

**Description :**

In character stuffing, each frame starts with the ASCII character sequence DLE STX and ends with the sequence DLE ETX.(where DLE is Data Link Escape, STX is Start of TeXt and ETX is End of TeXt.) This method overcomes the drawbacks of the character count method. If the destination ever loses synchronization, it only has to look for DLE STX and DLE ETX characters. If however, binary data is being transmitted then there exists a possibility of the characters DLE STX and DLE ETX occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used. The sender's data link layer inserts an ASCII DLE character just before the DLE character in the data. The receiver's data link layer removes this DLE before this data is given to the network layer. However character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary sized characters.

**Program :**

#include<stdio.h>

#include<conio.h>

void main()

{

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

char d[20];

printf("Enter the number of characters : ");

scanf("%d",&n);

printf("Enter the characters : ");

scanf("%s",d);

printf("The original data : %s\n",d);

printf("The transmitted data : dlestx",d);

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

{

j=i;

if(d[j]=='d')

{

if(d[++j]=='l')

{

if(d[++j]=='e')

{

printf("dle");

}

}

}

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

}

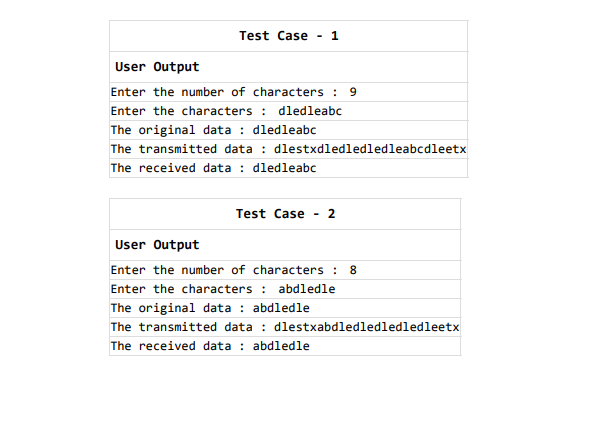
printf("dleetx");

printf("\nThe received data : %s",d);

printf("\n");

}

**Output :**

****

**Exp No : 2(a)**

**Aim :**

Write a program to implement on a data set of characters the CRC encoding algorithm.

**Description :**

CRC (Cyclic Redundancy Check) is used for encoding the given bits using the generated string. The encoded information is transmitted to the other end. For encoding we perform the exclusive operation

**Program :**

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char msg[20],key[10],keycpy[20],temp[20],quot[20],rem[20];

int msglen,keylen,i,j;

printf("Enter the frame string : ");

scanf("%s",msg);

printf("Enter the generator string : ");

scanf("%s",key);

strcpy(keycpy,key);

msglen=strlen(msg);

keylen=strlen(key);

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

{

msg[msglen+i]='0';

}

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

temp[i]=msg[i];

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

{

quot[i]=temp[0];

if(quot[i]=='0')

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

key[j]='0';

else

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

key[j]=keycpy[j];

for(j=keylen-1;j>0;j--)

{

if(temp[j]==key[j])

rem[j-1]='0';

else

rem[j-1]='1';

}

rem[keylen-1]=msg[i+keylen];

strcpy(temp,rem);

}

strcpy(rem,temp);

printf("The data to be transmitted is : ");

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

printf("%c",msg[i]);

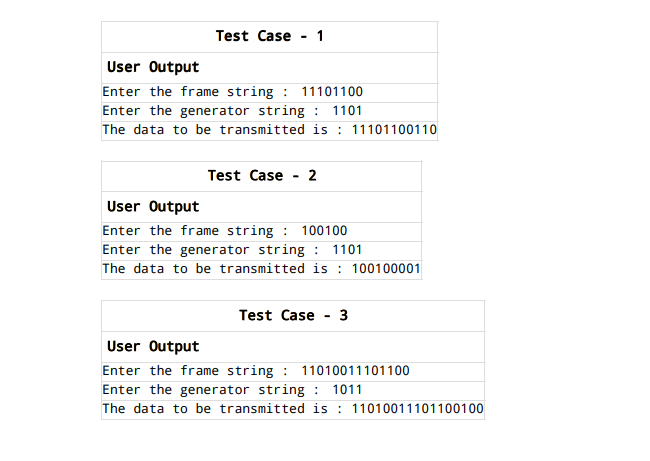
for(i=0;i<keylen-1;i++)

printf("%c",rem[i]);

printf("\n");

}

**Output :**

****

**Exp No : 2(b)**

**Aim :**

Write a program to implement on a data set of characters the CRC decoding algorithm

**Description :**

CRC (Cyclic Redundancy Check) is used for decoding the string which is encoded in the CRC encoding algorithm. Also we check for the errors that occur in the transmitted data. For this we use exclusive operation.

**Program :**

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

int p[20],g[20],i,j,k;

char cp[20],cg[20];

printf("Enter the received string : ");

scanf("%s",cp);

printf("Enter the polynomial string : ");

scanf("%s",cg);

for(i=0;i<strlen(cp);i++)

p[i]=cp[i]-'0';

for(i=0;i<strlen(cg);i++)

g[i]=cg[i]-'0';

i=0;

j=0;

while(1)

{

j=0;

while(p[j]!=1)

j++;

if(j>=strlen(cp)-(strlen(cg)-1))

break;

k=0;

for(i=j;i<j+strlen(cg);i++)

p[i]=p[i]^g[k];

}

for(i=0;i<strlen(cp);i++)

{

if(p[i]!=0)

break;

}

if(i!=strlen(cp))

{

printf("No error\n");

printf("Data received is : ");

for(i=0;i<strlen(cp)-(strlen(cg)-1);i++)

{

cp[i]=cp[i]-'0';

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

cp[i]=cp[i]-'1';

}

}

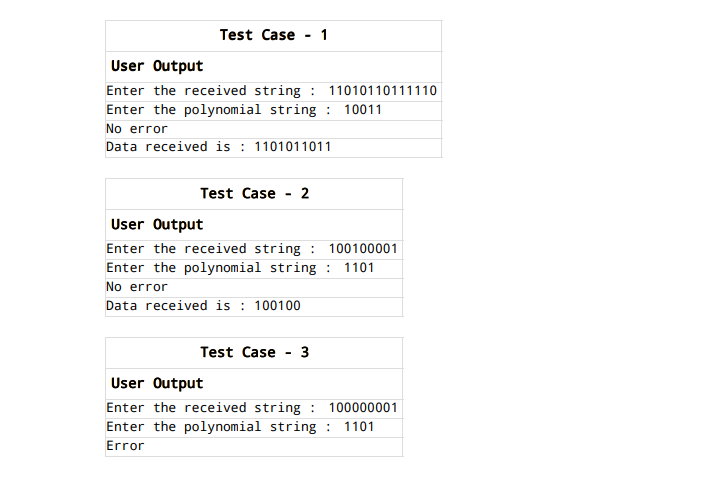
else

printf("Error");

printf("\n");

}

**Output :**

****

**Exp No : 3**

**Aim :**

Write a program on Implementation of Dijkstra's shortest path algorithm.

**Description :**

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y.

Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.
3. For the current node, consider all of its unvisited neighbours and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbour B has length 2, then the distance to B through A will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
4. When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

When planning a route, it is actually not necessary to wait until the destination node is "visited" as above: the algorithm can stop once the destination node has the smallest tentative distance among all "unvisited" nodes (and thus could be selected as the next "current").

**Program :**

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dij(int g[MAX][MAX],int n,int stnode);

int main()

{

int g[MAX][MAX],i,j,n,u;

printf("Enter number of vertices : ");

scanf("%d",&n);

printf("Enter the adjacency matrix\n");

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

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

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

printf("Enter the starting node : ");

scanf("%d",&u);

dij(g,n,u);

return 0;

}

void dij(int g[MAX][MAX],int n,int stnode)

{

int cost[MAX][MAX],dist[MAX],pred[MAX];

int vis[MAX],count,mindist,nextnode,i,j;

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

{

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

{

if(g[i][j]==0)

cost[i][j]=INFINITY;

else

cost[i][j]=g[i][j];

}

}

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

{

dist[i]=cost[stnode][i];

pred[i]=stnode;

vis[i]=0;

}

dist[stnode]=0;

vis[stnode]=1;

count=1;

while(count<n-1)

{

mindist=INFINITY;

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

{

if(dist[i]<mindist&&!vis[i])

{

mindist=dist[i];

nextnode=i;

}

}

vis[nextnode]=1;

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

{

if(!vis[i])

{

if(mindist+cost[nextnode][i]<dist[i])

{

dist[i]=mindist+cost[nextnode][i];

pred[i]=nextnode;

}

}

}

count++;

}

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

{

if(i!=stnode)

{

printf("Distance of node %d = %d\n",i,dist[i]);

printf("Path = %d",i);

j=i;

do

{

j=pred[j];

printf(" <- %d",j);

}while(j!=stnode);

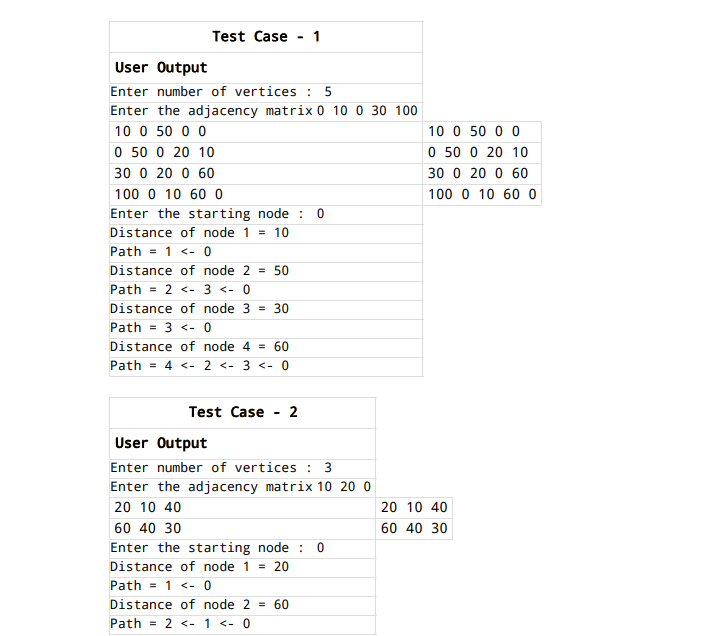
printf("\n");

}

}

}

**Output :**

****

**Exp No : 4**

**Aim :**

Take an example subnet graph with weights indicating delay between nodes. Now obtain routing table at each node using distance vector routing algorithm .

**Description :**

A router transmits its distance vector to each of its neighbours in a routing packet. Each router receives and saves the most recently received distance vector from each of its Neighbours.A router recalculates its distance vector when:

1. It receives a distance vector from a neighbour containing different information than before.
2. It discovers that a link to a neighbour has gone down.

**Program :**

#include<stdio.h>

struct node

{

unsigned dist[20];

unsigned from[20];

}rt[10];

int main()

{

int cost[10][10],n,i,j,k,count=0;

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

scanf("%d",&n);

printf("Enter the cost matrix\n");

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

{

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

{

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

cost[i][i]=0;

rt[i].dist[j]=cost[i][j];

rt[i].from[j]=j;

}

}

do

{

count=0;

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

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

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

if(rt[i].dist[j]>cost[i][k]+rt[k].dist[j])

{

rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];

rt[i].from[j]=k;

count++;

}

}while(count!=0);

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

{

Prin tf("For router %d\n",i+1);

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

{

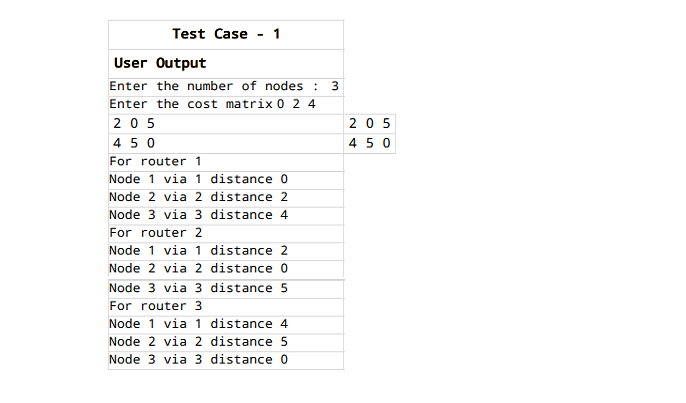
printf("Node %d via %d distance %d\n",j+1,rt[i].from[j]+1,rt[i].dist[j]);

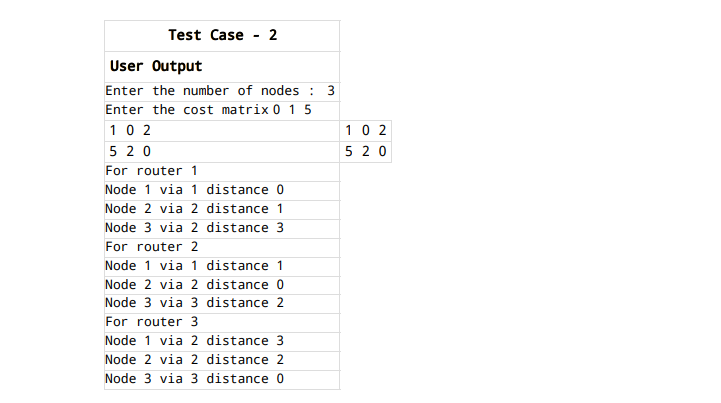
}

}printf("\n");

}

**Output :**

****

****

**Exp No : 5**

**Aim :**

Write a program to take an example subnet of hosts and obtain the broadcast tree for it.

**Description :**

Kruskal's algorithm is used to obtain the broadcast tree from the given subnet. This algorithm constructs a minimal spanning tree for a connected weighted graph G. Algorithm :

1. Select any edge of minimal value that is not a loop. This is the first edge of T.
2. Select any remaining edge of G having minimal value that does not from a circuit with the edges already included in T.
3. Continue step-2 until T contains n-1 edges where n is the number of vertices of G.

**Program :**

#include<stdio.h>

#include<conio.h>

int n,weight[10][10]={0},intree[10]={0},d[10]={0},whoto[10]={0};

void updatedistance(int target)

{

int i;

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

{

if((weight[target][i]!=0)&&(d[i]>weight[target][i]))

{

d[i]=weight[target][i];

whoto[i]=target;

}

}

}

void main()

{

int i,j,total=0,s;

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

scanf("%d",&n);

printf("Enter distance from\n");

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

{

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

{

if(i==j)

weight[i][j]=0;

else

{

printf("%c --> %c : ", 'A'+i,'A'+j);

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

}

}

}

printf("The Distance matrix is\n");

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

{

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

printf("%d ",weight[i][j]);

printf("\n");

}

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

d[i]=1000;

printf("Enter source from 0 to %d : ",n-1);

scanf("%d",&s);

printf("Broadcast tree for source node %c is\n",s+'A');

printf("Source\tDestination\n");

intree[s]=1;

updatedistance(s);

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

{

int min=-1;

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

{

if(!intree[i])

{

if((min==-1)||(d[min]>d[i]))

min=i;

}

}

printf("%c\t%c\n",whoto[min]+'A',min+'A');

intree[min]=1;

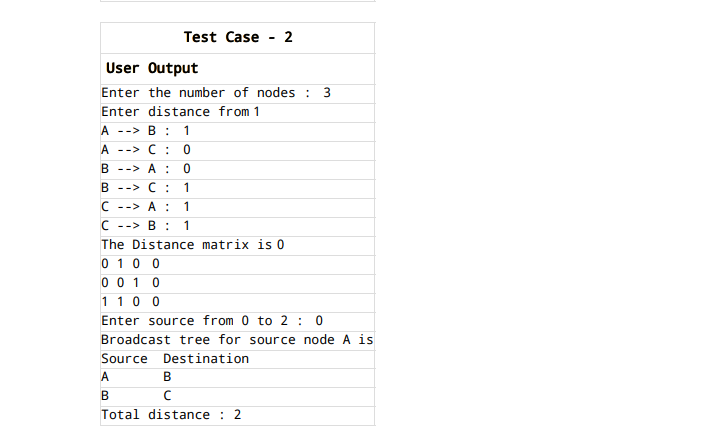
total+=d[min];

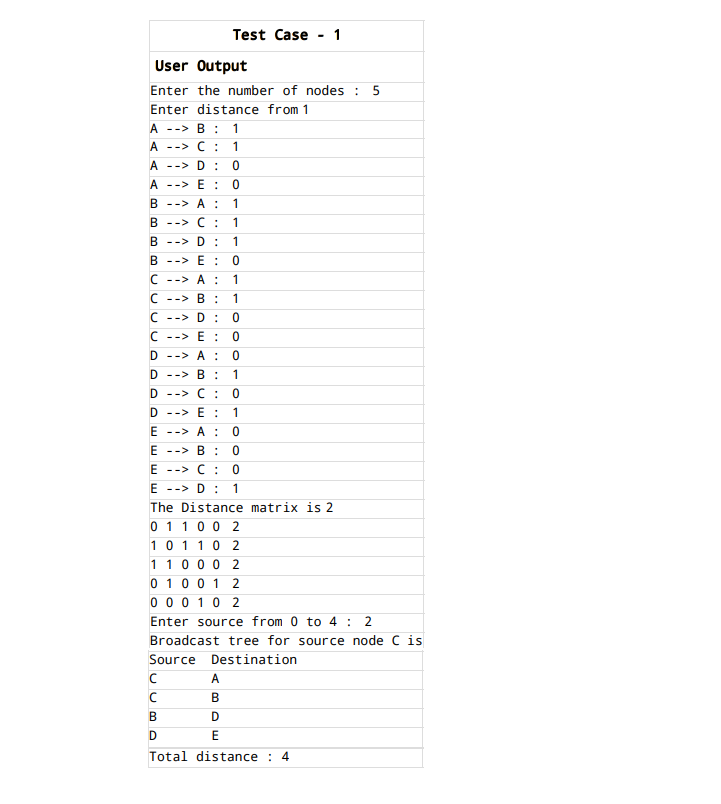
updatedistance(min);

}

printf("Total distance : %d\n",total);

}

**Output :**

****

**Exp No : 6**

**Aim :**

Implement hierarchical routing algorithm.

**Description :**

This is essentially a 'Divide and Conquer' strategy. The network is divided into different regions and a router for a particular region knows only about its own domain and other routers.

Thus, the network is viewed at two levels:

1. The Sub-network level, where each node in a region has information about its peers in the same region and about the region's interface with other regions. Different regions may have different 'local' routing algorithms. Each local algorithm handles the traffic between nodes of the same region and also directs the outgoing packets to the appropriate interface.
2. The Network Level, where each region is considered as a single node connected to its interface nodes. The routing algorithms at this level handle the routing of packets between two interface nodes, and is isolated from intra-regional transfer.

Networks can be organized in hierarchies of many levels; e.g. local networks of a city at one level, the cities of a country at a level above it, and finally the network of all nations.

In Hierarchical routing, the interfaces need to store information about:

1. All nodes in its region which are at one level below it.
2. Its peer interfaces.
3. At least one interface at a level above it, for outgoing packages

**Program :**

#include<stdio.h>

#include<conio.h>

struct full

{

char line[10],dest[10];

int hops;

}f[20];

void main()

{

int nv,i,min,minver;

char sv[20],temp;

printf("Enter number of vertices : ");

scanf("%d",&nv);

printf("Enter source vertex : ");

scanf("%s",sv);

printf("Enter full table for soutce vertex %s\n",sv);

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

scanf("%s%s%d",f[i].dest,f[i].line,&f[i].hops);

printf("HIERARCHICAL TABLE");

for(i=0;i<nv;)

{

if(sv[0]==f[i].dest[0])

{

printf("\n%s %s %d",f[i].dest,f[i].line,f[i].hops);

i++;

}

else

. {

min=1000;

minver=0;

temp=f[i].dest[0];

while(temp==f[i].dest[0])

{

if(f[i].hops<min)

{

. minver=i;

. }

i++;

}

. printf("\n%c %s %d",temp,f[minver].line,f[minver].hops);

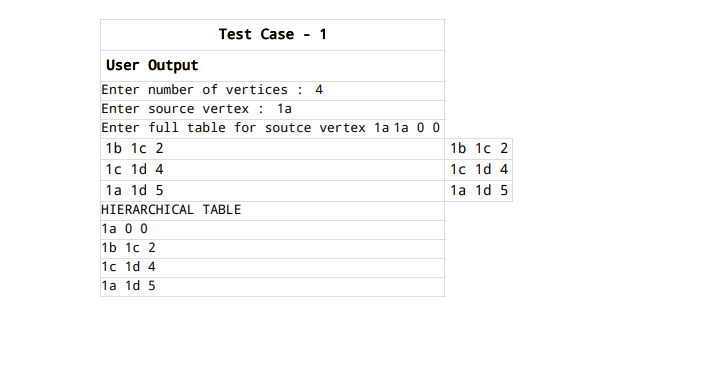
. }

}

printf("\n");

}

**Output :**

****

**Exp No : 7(a)**

**Aim :**

Write a program to implement caesar cipher substitution technique.

**Description :**

In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plain text is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence. The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions.

**Program :**

#include<stdio.h>

#include<string.h>

int main()

{

char msg[100],ch;

int i,key;

printf("Enter the string : ");

gets(msg);

printf("Enter the key value : ");

scanf("%d",&key);

printf("Original string : %s\n",msg);

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

{

ch=toupper(msg[i]);

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

{

ch=ch+key;

if(ch>'z')

{

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

}

msg[i]=ch;

}

else if(ch>='A' && ch<='Z')

{

ch=ch+key;

if(ch>'Z')

{

ch=ch-'Z'+'A'-1;

}

msg[i]=ch;

}

}

printf("Encrypted string : %s\n",msg);

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

{

ch=msg[i];

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

{

ch=ch-key;

if(ch<'a')

{

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

}

msg[i]=ch;

}

else if(ch>='A'&&ch<='Z')

{

ch=ch-key;

if(ch<'A')

{

ch=ch+'Z'-'A'+1;

}

msg[i]=ch;

}

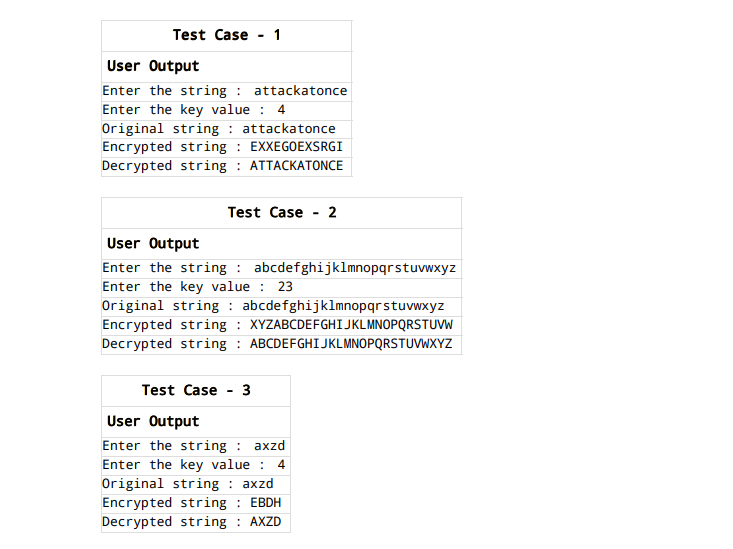
}

printf("Decrypted string : %s\n",msg);

return 0;

}

**Output :**

****

**Exp No : 7(b)**

**Aim :**

Write a program to implement rail fence cipher transposition technique.

**Description :**

A transposition cipher is one in which plain text symbols are rearranged (i.e., transposed or permuted) to produce cipher text. The method of transposition may be either mathematical or typographical in nature. The Rail fence cipher is a transposition cipher. It rearranges the plain text letters by drawing them in a way that they form a shape of the rails of an imaginary fence. To encrypt the message, the letters should be written in a zigzag pattern, going downwards and upwards between the levels of the top and bottom imaginary rails. The shape that is formed by the letters is similar to the shape of the top edge of the rail fence. Next, all the letters should be read off and concatenated, to produce one line of cipher text. The letters should be read in rows, usually from the top row down to the bottom one. The secret key is the number of levels in the rail. It is also a number of rows of letters that are created during encryption. This number cannot be very big, so the number of possible keys is quite limited.

**Program :**

#include<stdio.h>

#include<string.h>

void main()

{

int n,L,i,j,k=-1,row=0,col=0;

char a[40],b[40][40];

printf("Enter the plain text : ");

gets(a);

printf("Enter the value of n : ");

scanf("%d",&n);

L=strlen(a);

b[n][L];

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

{

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

{

b[i][j]='\n';

}

}

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

{

b[row][col++]=a[i];

if(row==0 || row==n-1)

k=k\*(-1);

row=row+k;

}

printf("The encrypted code : ");

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

{

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

{

if(b[i][j]!='\n')

{

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

}

}

}

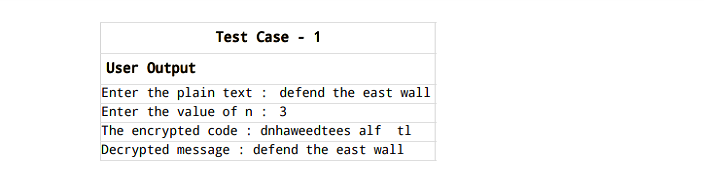
printf("\nDecrypted message : ");

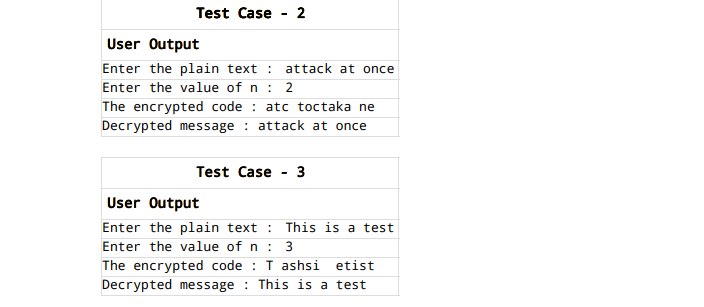
puts(a);

printf("\n");

}

**Output :**

****

****

**Exp No : 8**

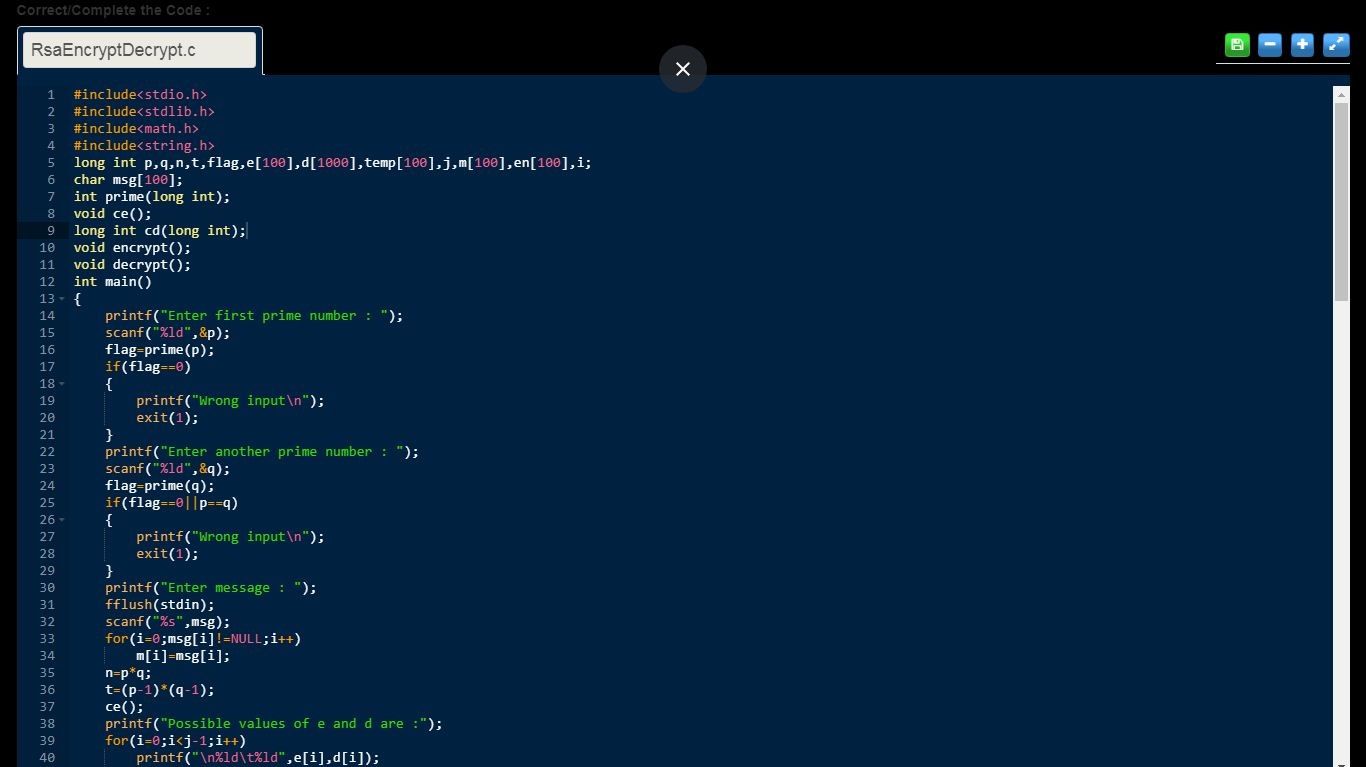
**Aim :**

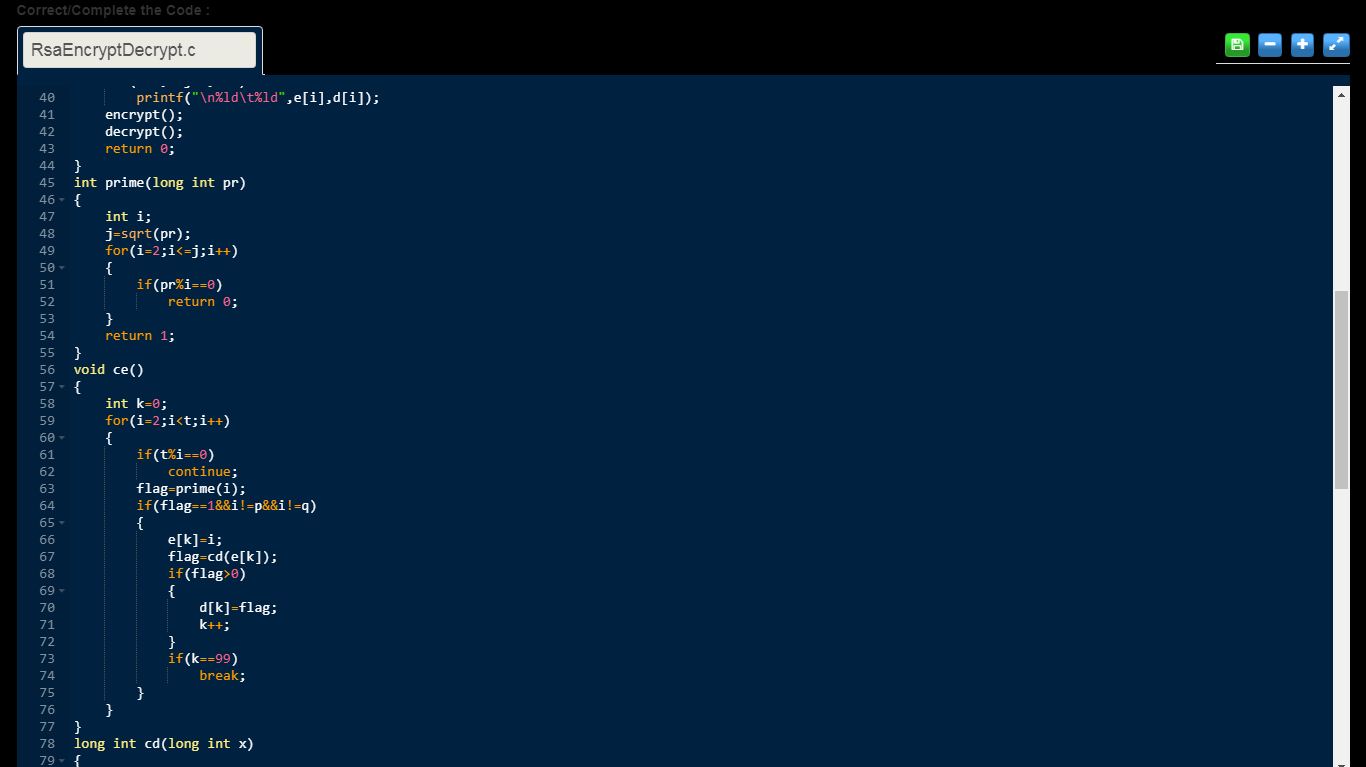
Write a program to implement RSA algorithm to encrypt a text data and decrypt the same.

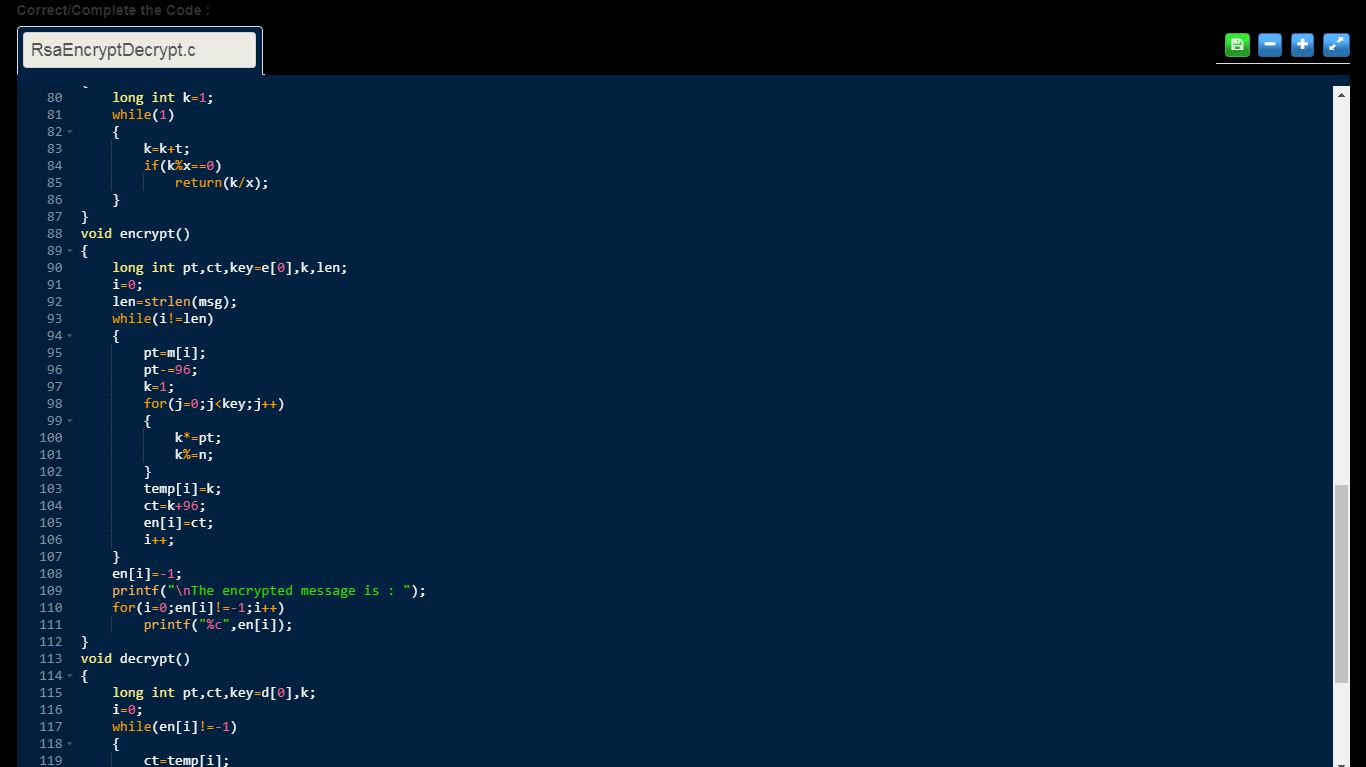
**Description :**

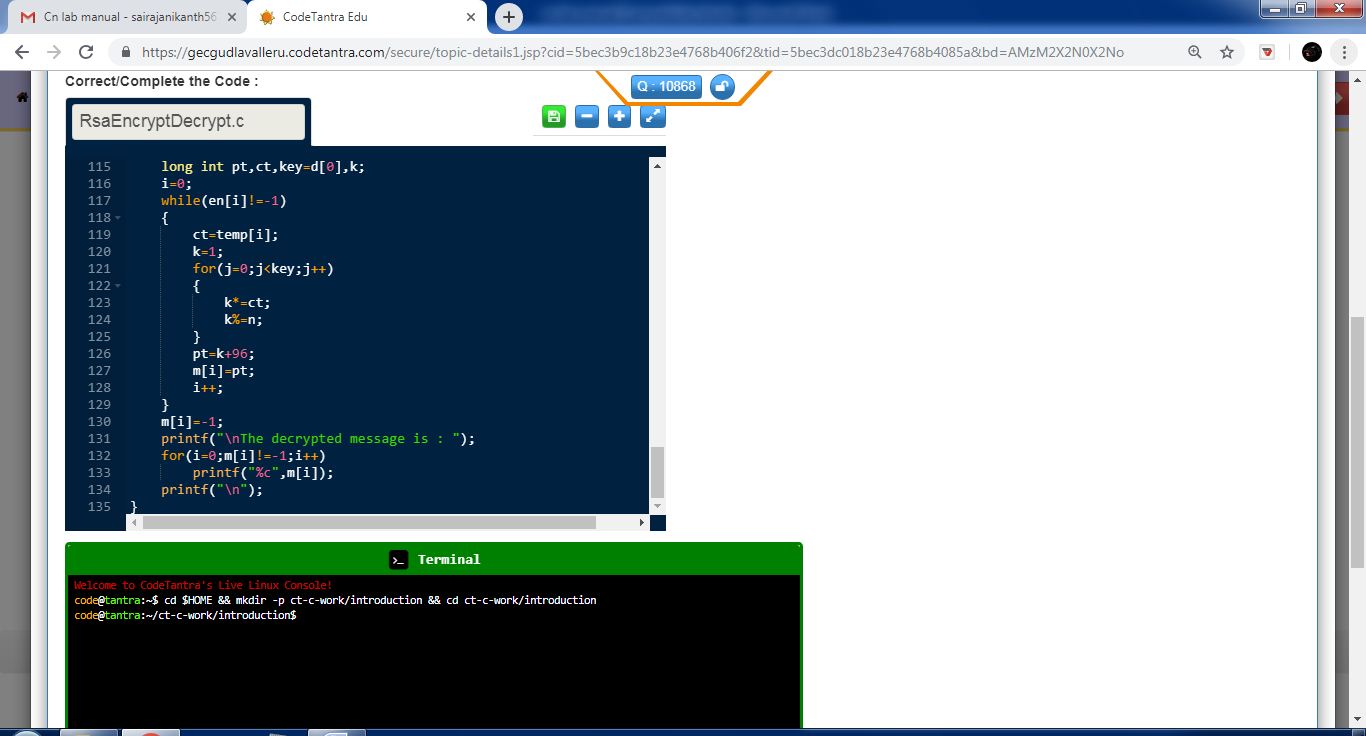
RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of the keys can be given to anyone. The other key must be kept private. The algorithm is based on the fact that finding the factors of a large composite number is difficult: when the integers are prime numbers, the problem is called prime factorization. It is also a key pair (public and private key) generator.

**Program :**

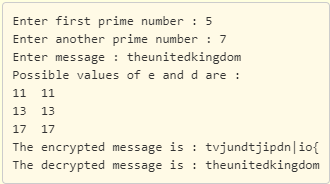
****

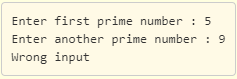
****

****

****

**Output :**

****

****

**Exp No : 9**

**Aim :**

Write a program to implement one of the error detection technique , simple parity check for even parity.

**Description :**

Blocks of data from the source are subjected to a check bit or parity bit generator form, where a parity of :

1 is added to the block if it contains odd number of 1’s, and

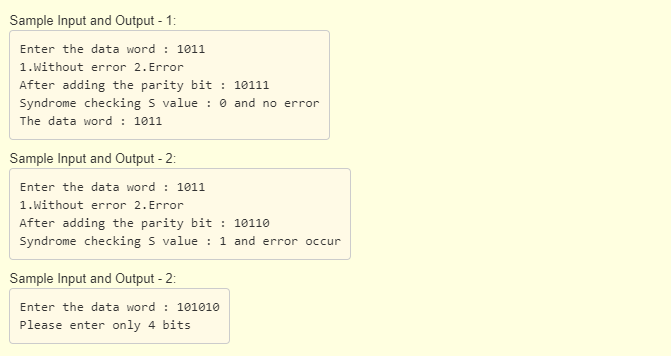
0 is added if it contains even number of 1’s

This scheme makes the total number of 1’s even, that is why it is called even parity checking.

**Program :**

****

**Output :**

****