## WRITE A C PROGRAM FOR A HDLC FRAME TO PERFORM

i. BIT STUFFING

ii. BYTE STUFFING

```
#include<stdio.h>
#include<string.h>
void main()
{
  int i,j,count=0;
  char str[100],dest[100]="";
  clrscr();
  printf("enter the bit string:");
  gets(str);
  for (i=0;i<strlen(str);)
  {
    count=0;
    for (j=i;j<(i+5)&&str[j]!='\0';j++)
    {
        if(str[j]=='1')
        {
           count++;
        }
    }
    if(count==5)
    {
        strcat(dest,"111110");
        i=i+5;
    }
    else
    {
         char temp[2];
         temp[0]=str[i];
         temp[1]='\0';
         strcat(dest,temp);
         i++;
    }
}
printf("After Stuffing\n");
puts(dest);
getch();
```

CHARACTER STUFFING (BYTE STUFFING)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
 char str[50],dest[50]=" ";
 int i;
 clrscr();
 printf("enter the character string:");
 gets(str);
 printf("\n original data:\n");
 puts(str);
 strcat(dest,"dlestx");
 for(i=0;i<strlen(str);)
 {
  if((str[i]=='d'&&str[i+1]=='l'&&str[i+2]=='e'))
  {
    strcat(dest,"dledle");
    i=i+3;
  }
  else
  {
    char temp[2];
    temp[0]=str[i];
    temp[1]='\0';
    strcat(dest,temp);
    i++;
  }
```

```
}
 strcat(dest,"dleetx");
 printf("\n after stuffing:\n");
 puts(dest);
  getchar();
}
```

## 2. DISTANCE VECTOR ALGORITHM TO FIND SUITABLE PATH FOR TRANSMISSION
(SHORTEST PATH)

```
#include<stdio.h>
#include<stdlib.h>
int n,cost[10][10],via[10][10],i,j,k;
int main()
{
 clrscr();
 printf("enter the num of nodes(less than 10)\n");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
       printf("\n the cost from %d:(999 for no connection)\n",i+1);
       for(j=0;j<n;j++)
       {
             if(i!=j)
             {
```

```
                printf("to %d :",j+1);
                scanf("%d",&cost[i][j]);
            }
            else
            cost[i][j]=0;
    }
    }
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    via[i][j]=i;
    for(i=0;i<n;i++)
    {
    for(j=0;j<n;j++)
    {
     if(cost[i][j]<999)
{
        for(k=0;k<n;k++)
         {
           if(cost[i][k]>cost[i][j]+cost[j][k])
           {
               cost[i][k]=cost[i][j]+cost[j][k];
               via[i][k]=j;
           }
         }
    }
  }
}
for(i=0;i<n;i++)
{
 printf("the prefered path from %d to:\n",i+1);
 for(j=0;j<n;j++)
 {
  if (cost[i][j]!=999)
  printf("%d:%d via %d \n",j+1,cost[i][j],via[i][j]+1);
  else
  printf("%d: no connection \n, j+1");
  }
  }
  getch();
  }
```

## 3. IMPLEMENT DIJIKSTRA'S ALGORITHM TO COMPUTE SHORTEST PATH

```
#include <stdio.h>
#include<conio.h>
void dijikstras (int cost[10][10],int dist[10],int n, int v)
{
 int i,u,w,count,flag[10],min;
 for(i=1;i<=n;i++)
 {
  flag[i]=0;
  dist[i]=cost[v][i];
 }
  flag[v]=1;
  dist[v]=0;
  count=2;
  while(count<n)
  {
   for(i=1,min=999;i<=n;i++)
   {
    if (dist[i]<min &&! flag[i])
    {
```

```
        min=dist[i];
        u=i;
       }
    }
    flag[u]=i;
    count++;
    for(w=1;w<=n;w++)
    {
     if(dist[u]+cost[u][w]<dist[w] &&! flag[w])
     dist[w]=dist[u]+cost[u][w];
    }
   }
  }

  int main()
  {
   int n,cost[10][10],source,i,j,dist[10];
   clrscr();
   printf("enter the number of vertices\n");
   scanf("%d",&n);
   printf("enter the cost matrix \n");
   for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
   {
    scanf("%d",&cost[i][j]);
    if(cost[i][j]==0)
    cost[i][j]=999;
  }
  printf("source\n");
  scanf("%d",&source);
  dijikstras(cost,dist,n,source);
  printf("vertex \t destination \tcost \n");
  for(i=1;i<=n;i++)
  if(source!=i)
  printf("%d\t%d\t%d\n",source,i,dist[i]);
  getch();
  //getchar();
  }
```

4. USE CRC-CCITT POLYNOMIAL TO OBTAIN CRC CODE

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int error;
char i[200],o[200],g[200];
int crc(char i[200],char o[200],int mode);
int void main()
{
  char r[200];  → clrscr();
  printf("\n enter the message in binary");
  scanf("%s",i);
  crc(i,o,1);
  printf("\n the crc code is %s%s",i,o+strlen(i));
  printf("\n enter the recieved message");
  scanf("%s",r);
  if(!crc(r,o,0))
  printf("\n error free message\n");
  else
  printf("\n error in message\n");
  getch();
  return 0;
  }
```

```
int crc (char i[200], char o[200], int mode)
{
int j,k;
char g[200]={"1101"};→( g(x) )
strcpy(o,i);
if(mode)
strcat(o,"000"); ⟶ 4 bit → 3 zeros | 5 bit → 4 zeros
for(j=0;j<strlen(i);j++)
if(o[j]=='1')
for(k=0;k<strlen(g);k++)
if(o[j+k]=='1'&& g[k]=='1')||(o[j+k]=='0'&& g[k]=='0')) → XOR operation
o[j+k]='0';
else
o[j+k]='1';
for(j=0;j<strlen(o);j++)
{
  error=0;
  if(o[j]=='1')
  {
     error=1;
     break;
     }
     }
     if(error==1)
     return 1;
     else
     return 0;
     }
```

1101 → n = 5 → n − 1 = 0 to 3

CRC  g(x) → generated polynomial

P(x) → msg polynomial

g(x) = 1101  P(x) = 1001

g(x) − 1 = 0 to 3  P(x) appended

XOR truth table

0 0 } → 0
1 1 }

0 1 } → 1
1 0 }

0 0 } → 0
1 1 }

0 1 } → 1
1 0 }

```
          1 1 1 1
      ┌──────────
1101  ) 1 0 0 1 0 0 0 0
XOR     1 1 0 1
        ---------
        0 1 0 0 0
          1 1 0 1
          -------
        0 1 0 1 0
            1 1 0 1
            -------
        0 1 1 1 0
              1 1 0 1
              -------
        0 0 1 1 1
```

1101 ⟌ 1001011



R = 1 = error in msg
R = 0 = error free msg

## 5. CONGESTION CONTROL USING LEAKY BUCKET ALGORITHM

```
#include<stdio.h>
#include<stdlib.h>
#define bucketsize 512
void bktoutput(int a, int b)
{
if(a>bucketsize)
printf("\n\t\tBucket overflow,Hence discarded");
else
{
 while(a>b)
  {
        printf("\n\t\t%d bytes outputted",b);
        a-=b;
  }
  if(a>0)
  printf("\n\t\t%d bytes sent\t",a);
  printf("bucket output successful\n\n");
  }
}
```

```
void main()
{
int op,pktsize,n;
int size[10],i;
clrscr();
printf("\n enter the output rate:");
scanf("%d",&op);
printf("\n enter the number of packets");
scanf("%d",&n);
printf("\n enter the size of packets");
for (i=1;i<=n;i++)
        scanf("%d",&size[i]);
        for(i=1;i<=n;i++)
        {
                printf("\n packet no %d",i);
                bktoutput(size[i],op);
        }
getch();
}
```

6. IMPLEMENTATION OF STOP AND WAIT PROTOCOL AND SLIDING WINDOW PROTOCOL

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int i,j,noframes=0,x,x2;
clrscr();
i=1;
j=1;
printf("number of frames is ");
scanf("%d",&noframes);
while(noframes>0)
{
printf("\nsending frames is %d",i);
x=rand()%15;
if(x%5==0)
{
```

```
for(x2=1;x2<3;x2++)
{

printf("\n waiting for %d seconds\n",x2);
sleep(x2);
}
printf("\n sending frames %d\n",i);
}
printf("\n ack for frame %d\n",j);
noframes=noframes-1;
i++;
j++;
}
printf("\n end of stop and wait protocol\n");
getch();
}
```

SLIDING WINDOW PROTOCOL

```
#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; getch();
}
```