1. **Write a program for error detection code using CRC**

```
#include<stdio.h>
int main( )
{
        int i,j,k=0;
        int flag=1,a[16],g[16],r[20],div[16],n,m;
        printf("\n enter the degree of generator");
        scanf("%d",&n);
        printf("enter the generator:");
        for( i=0; i<=n; i++ )
        scanf("%d",&g[i]);
        printf("enter the degree of frames:");
        scanf("%d",&m);
        printf("enter the frame");
        for( i=0;i<=m;i++)
        scanf("%d",&a[i]);
        if(m<n||(g[0]&&g[n])==0)
        {
        printf("not a proper generator\n");
        }
        for(i=m+1;i<=m+n;i++)
        a[i]=0;
        for(j=0;j<=n;j++)
        r[j]=a[j];
        for(i=n;i<=m+n;i++)
        {
        if(i>n)
        {
```

```
for(j=0;j<n;j++)
r[j]=r[j+1];
r[j]=a[i];
}
if(r[0])
div[k++]=1;
else
{
div[k++]=0;
continue;
}
for(j=0;j<=n;j++)
r[j]=r[j]^g[j];
}
printf("\n quetient is\n");
for(j=0;j<k;j++)
printf(" %d",div[j]);
printf("\n remainder is");
for(i=1;i<=n;i++)
printf("%d",r[i]);
printf("\n transmitted frame is");
for(i=m+1,j=1;i<=m+n;i++,j++)
a[i]=r[j];
for(i=0;i<=m+n;i++)
printf("%d",a[i]);
printf("\n");
printf("\n enter the degree of frame");
scanf("%d",&m);
printf("\n enter the frame");
```

```
for(i=0;i<=m;i++)
scanf("%d",&a[i]);
for(j=0;j<=n;j++)
r[j]=a[j];
k=0;
for(i=n;i<=m;i++)
{
if(i>n)
{
for(j=0;j<n;j++)
r[j]=r[j+1];
r[j]=a[i];
}
if(r[0])
div[k++]=1;
else
{
div[k++]=0;
continue;
}
for(j=0;j<=n;j++)
r[j]=r[j]^g[j];
}
printf("\n quetient is\n");
for(j=0;j<k;j++)
printf("%d",div[j]);
printf("\n remainder is");
for(i=1;i<=n;i++)
printf("%d",r[i]);
```

```
for(i=1;i<=n;i++)
{
if(r[i])
flag=0;
}
if(flag)
printf("\n no error");
else
printf("\n error");
}
```

Output:

2. **Write a program for frame sorting technique used in buffers**.

```
#include<stdio.h>
#include<string.h>
struct frame
{
         int seq;
        int len;
        int flag;
        char data[10];

 }n[20],m[20],temp;
char str[100];
int count=0;
void frames( )
{
        int i,j,s,size,total=0;
        s=strlen(str);
while( total<s )
{
         size=rand()%10+1;
        n[count].seq=count+1;
        n[count].len=size;
        n[count].flag=0;
if((total+size)<s)
 {
for(i=total,j=0;j<size;i++,j++)
        n[count].data[j]=str[i];
        total+=size;
 }
```

```
else
  {
        n[count].len=s-total;
        for(j=0;j<n[count].len;j++)
        n[count].data[j]=str[total++];
  }
count+=1;
}
printf("\n show the packets;\n\n");
for(i=0;i<count;i++)
  {
        printf("\t%d:%d\t",n[i].seq,n[i].len);
        for(j=0;j<n[i].len;j++)
        printf("%c",n[i].data[j]);
        printf("\n");
  }
}
void trans( )
{
        int i,j;
        int c=0;
while(c<count)
 {
        i=rand()%count;
        if(n[i].flag==0)
  {
        m[c++]=n[i];
        n[i].flag=1;
  }
 }
printf("\n\n show the random packets\n\n");
for(i=0;i<count;i++)
 {
        printf("\t%d:%d\t",m[i].seq,m[i].len);
        for(j=0;j<m[i].len;j++)
        printf("%c",m[i].data[j]);
        printf("\n");
 }
}
void sort(  )
{
        int i,j;
        for(i=0;i<count;i++)
        for(j=i+1;j<count;j++)
if(m[i].seq>m[j].seq)
  {
        temp=m[i];
        m[i]=m[j];
        m[j]=temp;
  }
printf("\n\n show the sequenced packets:\n\n");
for(i=0;i<count;i++)
  {
        printf("\t%d:%d\t",m[i].seq,m[i].len);
        for(j=0;j<m[i].len;j++)
        printf("%c",m[i].data[j]);
        printf("\n");
   }
```

```
}
main(   )
{
        system("clear");
        printf("enter the data");
        scanf("%s",(str));
        frames();
        trans();
        sort();
}
```

Output:

3. Write a program for distance vector algorithm to find suitable path for transmission.

```
#include<stdio.h>
#include<conio.h>
struct rtable                    /*structure of routing table*/
        {
        int dist[20];
        int nextnode[20];
        }table[20];
int cost[20][20],adj[20][20],d[20];
int n,node;
void distvector();
void main()
{
int i,j,ch;
clrscr();
printf("enter the no. of nodes \n");
scanf("%d",&n);
printf("enter the cost matrix");   /*cost matrix */
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(i!=j&&cost[i][j]==0)
cost[i][j]=999;
if(i==j||cost[i][j]==999) /*finding neighbours for all the nodes*/
adj[i][j]=0;
else
```

```
adj[i][j]=1;
}
distvector();
printf("enter the node whose rout table is to be found  \n");
scanf("%d",&node);
printf("enter the delay to neighbouring node \n");
for(i=1;i<=n;i++)
if(adj[node][i]==1) /*showing the routing table of the neighbouring node*/
{
printf("node-->%d=",i);
scanf("%d",&cost[node][i]);
printf("delay of %d node to all other node is:\n",i);
for(j=1;j<=n;j++)
printf("%d\n",table[i].dist[j]);
}
distvector();      /*applying distance vector algorithm*/
printf("the new routing table for node %d is:\n",node);
for(i=1;i<=n;i++)
printf("%d-->%d\n",table[node].dist[i],table[node].nextnode[i]);
printf("\nshortest path from %d to other node is\n",node);
for(i=1;i<=n;i++)
printf("%d->%d=%d\n",node,i,table[node].dist[i]);
getch();
}
void distvector()
{
int i,j,k,count;
for(i=1;i<=n;i++)
{
```

```
for(j=1;j<=n;j++)
{
table[i].dist[j]=cost[i][j];/*assigning distance from node i to node j*/
table[i].nextnode[j]=j;//initialising the line
}
}
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
for(k=1;k<=n;k++)
{
if(table[i].dist[j]>cost[i][k]+table[k].dist[j]) //check the shortest distance to reach
j from i
{
table[i].dist[j]=table[i].dist[k]+table[k].dist[j];//if we can reach j from i using k
then set the path through k
table[i].nextnode[j]=k; //make the line to reach j as k
}
}
}
}
}
```

Output:

4. **Using TCP/IP sockets,write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.**

/*server side source code*/

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<string.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<sys/wait.h>
#include<signal.h>
#define MYPORT 6490
#define BACKLOG 10
int main(void)
{
int sockfd,fp,new_fd;
struct sockaddr_in my_addr,their_addr;
int sin_size,i=0;
int yes=1;
char buf1[20],buf2[20000];
if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
```

```
perror("socket");
exit(1);
}
if(setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int))==-1)
{
perror("setsockopt");
exit(1);
}
my_addr.sin_family=AF_INET;
my_addr.sin_port=htons(MYPORT);
my_addr.sin_addr.s_addr=INADDR_ANY;
memset(&(my_addr.sin_zero),'\0',8);
if(bind(sockfd,(struct sockaddr *)&my_addr,sizeof(struct sockaddr))==-1)
{
perror("bind");
exit(1);
}
if(listen(sockfd,BACKLOG)==-1)
{
perror("listen");
exit(1);
}
printf("\nServer is online!!!!\n server waiting for the client\n");
sin_size=sizeof(struct sockaddr_in);
if((new_fd=accept(sockfd,(struct sockaddr *)&their_addr,&sin_size))==-1)
{
 perror("accept");
 exit(0);
}
```

```
printf("\n server got connection from %s\n",inet_ntoa(their_addr.sin_addr));
recv(new_fd,&buf1,sizeof(buf1),0);
printf("file request is %s\n",buf1);
if((fp=open(buf1,O_RDONLY))<0)
{
printf("File not found\n");
strcpy(buf2,"file not found");
}
else
{
printf("SERVER:%s found and ready to transfer\n",buf1);
read(fp,&buf2,20000);
close(fp);
}
send(new_fd,&buf2,sizeof(buf2),0);
close(new_fd);
close(sockfd);
printf("\n transfer successful\n");
printf("\n");
return 0;
}
```

/*client side source code*/

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<fcntl.h>
#include<netdb.h>
#include<errno.h>
#define PORT 6490

int main()
        {
        int i=0,sockfd;
        char buf1[40],buf2[20000];
        struct sockaddr_in their_addr;
        if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
                {
                        perror("socket");
                        exit(1);
                }
        their_addr.sin_family=AF_INET;
        their_addr.sin_port=htons(PORT);
        their_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
        memset(&(their_addr.sin_zero),'\0',8);
        if(connect(sockfd,(struct sockaddr *)&their_addr,sizeof(struct
sockaddr))==-1)
```

```
          {
          perror("connnect");
                    exit(1);
          }
printf("client is online\n");
printf("\n client:enter the filename to be displayed");
scanf("%s",buf1);
send(sockfd,buf1,sizeof(buf1),0);
if(recv(sockfd,buf2,sizeof(buf2),0)==1)
          {
                    perror("recv");
                    exit(1);
          }
else
          {
                    printf("\n displaying the contents of %s",buf1);
                    printf("\n %s\n",buf2);
          }
close(sockfd);
return 0;
}
```

Output:

5. **Implement the TCP/IP sockets, write a program as message queues or FIFOs as IPC channels.**

```
/*server side source code*/
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<string.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#define FIFO1_NAME "Server_fifo"
#define FIFO2_NAME "Client_fifo"
int main()
{
        char p[100],f[100],c[300];
        int num,num2,f1,fd,fd2;
        mknod(FIFO1_NAME,S_IFIFO | 0666,0);
        mknod(FIFO2_NAME,S_IFIFO | 0666,0);
printf("\n server is online!!!...\n");
        fd=open(FIFO1_NAME,O_RDONLY);
        printf("\n client is online!\n");
        while(1)
{
        if((num=read(fd,p,100))==-1)
        perror("read error\n");
else
{
```

```
p[num]='\0';
if((f1=open(p,O_RDONLY))<0)
{
printf("\nserver %s is not found",p);
exit(1);
}
else
{
printf("Server %s found!\n transferring the content\n",p);
stdin=fdopen(f1,"r");
while(!feof(stdin))
{
if(fgets(c,300,stdin)!=NULL)
{
fd2=open(FIFO2_NAME,O_WRONLY);
if(num2=write(fd2,c,strlen(c))==-1)
perror("Transfer error");
}
else
perror("read");
}
printf("Server transfer completed\n");
exit(1);
}
}
}
return 1;
}
```

/*client side source code*/

```
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<string.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#define FIFO1_NAME "Server_fifo"
#define FIFO2_NAME "Client_fifo"
int main()
{
        char p[100],f[100],c[300];
        int num,num2,f1,fd,fd2;
        mknod(FIFO1_NAME,S_IFIFO | 0666,0);
        mknod(FIFO2_NAME,S_IFIFO | 0666,0);
        printf("\n waiting for server...\n");
        fd=open(FIFO1_NAME,O_WRONLY);
        printf("\n server online!\nclient:enter the path\n");
        while(gets(p) , !feof(stdin))
{
        if((num=write(fd,p,strlen(p)))==-1)
        perror("write error\n");
else
{
        printf("\n waiting for reply....\n");
        fd2=open(FIFO2_NAME,O_RDONLY);
if((num2=read(fd2,c,300))==-1)
perror("transfer error!\n");
else
{
printf("file received!displaying the contents:\n");
if(fputs(c,stdout)==EOF)
perror("print error");
exit(1);
}
}
}
```

Output:

6. Write a program for simple RSA algorithm to encrypt and decrypt the data.

6. a)

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
long int e,d,n;
long int val[50];
char decode(long int ch)
{
 int i;
 long int temp=ch;
 for(i=1;i<d;i++)
 ch=(temp*ch)%n;
 return ch;
}
int gcd(long int a,long int b)
{
 long int temp;
 while(b!=0)
 {
  temp=b;
  b=a%b;
  a=temp;
 }
return a;
}
int prime(int a)
```

```
{
 int i;
 for(i=2;i<a;i++)
 {
  if((a%i)==0)
  return 0;
 }
return 1;
}
int encode(char ch)
{
 int i;
 long int temp;
 temp=ch;
 for(i=1;i<e;i++)
 temp=(temp*ch)%n;
 return temp;
}
int main()
{
int i;
long int p;
long int q,phi,c[50];
char text[50],ctext[50];
system("clear");
printf("\nEnter the text to be encoded\n");
scanf("%s",text);
do
{
```

```
p=rand()%30;
}while(!prime(p));

do
{
q=rand()%30;
}while(!prime(q));
n=p*q;
phi=(p-1)*(q-1);
printf("\np=%d\tq=%d\tn=%d\tphi=%d\n",p,q,n,phi);
do
{
e=rand()%phi;
}while(!gcd(e,phi));

do
{
d=rand()%phi;
}while(((d*e)%phi)!=1);
printf("\n***************Encoding Message***************\n");
sleep(3);
for(i=0;text[i]!='\0';i++)
val[i]=encode(text[i]);
val[i]=-999;

printf("Encode Message:\n");
for(i=0;val[i]!=-999;i++)
printf("%ld",val[i]);
printf("\n**************Decoding encrypted
Message******************\n");
sleep(3);
for(i=0;val[i]!=-999;i++)
ctext[i]=decode(val[i]);
ctext[i]='\0';
printf("Decoded message is:%s\n\n",ctext);
}
```

Output:

6. b)

```
#include<stdio.h>
#include<math.h>
double min(double x, double y)
 {
      return(x<y?x:y);
 }

double max(double x, double y)
 {
      return(x>y ? x:y);
 }

double gcd(double x, double  y)
 {
      if(x==y)
        return (x);
      else
        return(gcd(min(x,y),max(x,y)-min(x,y)));
 }


long double modexp(long double a,long double x,long double n)
 {
      long double r=1;
      while(x>0)
       {
        if((int)(fmodl(x,2))==1)
        {
         r=fmodl((r*a),n);
        }
        a=fmodl((a*a),n);
        x/=2;
       }
      return(r);
 }

main()
 {
      long double p,q,phi,n,e,d, ms,es,ds;
      system("clear");
      do{
```

```
    printf("\nEnter prime numbers P and Q : ");
    scanf("%Lf %Lf",&p,&q);
   }while(p==q);
  n=p*q;
  phi=(p-1)*(q-1);
  do{
    printf("\nEnter prime value of e : ");
    scanf("%Lf",&e);
   }while((gcd(e,phi)!=1) && e>phi);
  for(d=1;d<phi;++d)
  {
   if(fmod((e*d),phi)==1)
   break;
  }
  printf("\nD within main = %Lf",d);
  printf("\nEnter the message : ");
  scanf("%Lf",&ms);
  es=modexp(ms,e,n);
  ds=modexp(es,d,n);
  printf("\nOriginal message : %Lf",ms);
  printf("\nEncrypted message : %Lf",es);
  printf("\nDecrypted message : %Lf\n\n",ds);
  return(0);
}
```

Algorithm

1. Select two prime numbers P and Q
2. Calculate $n=P \times Q$
3. Calculate $\phi(n) = (P - 1) \times (Q - 1)$
4. Select e such that, e is relatively prime to $\phi(n)$ and less than n

   $\gcd(\phi(n),e)=1$
5. Determine d such that $de \equiv 1 \pmod{\phi(n)}$ and $d < \phi(n)$ $\gcd(\phi(n),d)=1$ d can be calculated using extended Euclid's Algorithm

   Public key PU = { e , n}

   Private key PR = { d , n }
6. $C=M^e \bmod n$, $M = C^d \bmod n$

   Encryption Decryption

Relatively prime – the two integers share no common positive factors except 1

Example:

1. Select P and Q, P=17 and Q=11
2. $n= P \times Q = 17 \times 11 = 187$
3. $\phi(n) = (P - 1) \times (Q - 1) = 16 \times 10 = 160$
4. Select e, $e = 7$
5. Determine d, $de \equiv 1 \bmod 160$

   $23 \times 07 \equiv 1 \bmod 160$

   $161 \equiv 1 \bmod 160 \rightarrow d=23$

PU={7,187}, PR={23,187}

Encryption:

6. $C=M^e \bmod n$ Given M=88 [M→ Plain Text Message]

$88^7 \bmod 187$

$= [(88^4 \bmod 187) \times(88^2 \bmod 187) \times(88^1 \bmod 187)] \bmod 187$

$88^1 \bmod 187 = 88$

$88^2 \bmod 187=7744 \bmod 187=77$

[In Calculator 7744 ÷ 187 = 41.41176471 – 41(Integer part value)]

$=0.4117647\times187=77$

$88^4 \bmod 187=132$

$88^7 \bmod 187= [88 \times 77 \times 132] \bmod 187 = 11$

∴ cipher text **<u>C=11</u>**

Decryption:

7. $M = C^d \bmod n$

$11^{23} \bmod 187$

$[(11^1 \bmod 127) \times(11^2 \bmod 187) \times(11^4 \bmod 187) \times(11^8 \bmod 127) \times(11^8 \bmod 127)] \bmod 187$

$11^1 \bmod 187 =11$, $11^2 = 121$, $11^4 = 55$, $11^8 = 33$

$11^{23} \bmod 127 = [11\times 121\times 55\times 33\times33] \bmod 187 = 88$

7. **Write a program for Hamming Code generation for error detection and correction**.

Encoding

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

char d1[5],d2[5],d3[5],d4[5];
char gmatrix[4][8];
char p1[5],p2[5],p3[5];
char data[5];
int encoded[8];

int con(char x);

int main()
{
        int i,j;
        system("clear");
        printf("\n Program for Hamming Code Implementation-Encoding\n");
        printf("Enter 4 data bits\n");
        scanf("%s",data);
        for(i=0;i<4;i++)
        {
                d1[i]=d2[i]=d3[i]=d4[i]='0';
                p1[i]=p2[i]=p3[i]='1';
        }
        printf("----------------------------\n");
```

```
d1[0]='1';
d2[1]='1';
d3[2]='1';
d4[3]='1';
p1[0]='0';
p2[1]='0';
p3[2]='0';
/*printf("%s\n",d1);
Printf("%s\n",d2);
printf("%s\n",d3);
printf("%s\n",d4);
printf("%s\n",p1);
Printf("%s\n",p2);
printf("%s\n",p3);*/
for(i=0;i<4;i++)
    gmatrix[i][0]=p1[i];
for(i=0;i<4;i++)
    gmatrix[i][1]=p2[i];
for(i=0;i<4;i++)
    gmatrix[i][2]=p3[i];
for(i=0;i<4;i++)
    gmatrix[i][3]=d1[i];
for(i=0;i<4;i++)
    gmatrix[i][4]=d2[i];
for(i=0;i<4;i++)
    gmatrix[i][5]=d3[i];
for(i=0;i<4;i++)
    gmatrix[i][6]=d4[i];
printf("\n generator matrix\n");
```

```
        for(i=0;i<4;i++)
        printf("%s\n",gmatrix[i]);

        for(i=0;i<7;i++)
           for(j=0;j<4;j++)
                encoded[i]=encoded[i]+con(data[j]*con(gmatrix[j][i]));
        puts("encoded");
        for(i=0;i<7;i++)
        {
                encoded[i]=encoded[i]%2;
                printf("%i",encoded[i]);
        }
        puts("");
        return 0;
}
int con(char x)
{
        if(x=='1')
                return 1;
        else
                return 0;
}
```

Decoding

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int hmatrix[][7]={  1,0,0,0,1,1,1,
                    0,1,0,1,0,1,1,
                    0,0,1,1,1,0,1
                  };
int edata[7],syndrome[3],errdig;
int main()
{
 int i,j;
 system("clear");
 printf("Enter the Encoded bits\n");
 for(i=0;i<7;i++)
 scanf("%d",&edata[i]);
 for(i=0;i<3;i++)
  for(j=0;j<7;j++)
   syndrome[i]+=edata[j]*hmatrix[i][j];
 for(i=0;i<3;i++)
  syndrome[i]%=2;
 errdig=3*syndrome[0]+2*syndrome[1]+1*syndrome[2];
 if(errdig==0)
  printf("Error free data\n");
 else
 {
  printf("Error in bit no %d---%d\n",errdig,edata[errdig]);
// errdig--;
 if(edata[errdig]==0)
 edata[errdig]=1;
 else
  edata[errdig]=0;
}
 for(i=3;i<7;i++)
  printf("%d",edata[i]);
 puts(" ");
}
```

Output:

8. Write a program for congestion control using Leaky bucket algorithm.

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

int t_rand(int a)
{
        int rn;
        rn=random()%10;
        rn=rn%a;
        if(rn==0)
        rn=1;
          return(rn);
}
main()
{
    int  i,j,clk,packets[5],o_rate,i_rate,b_size,p_remain,p_sz,p_sz_rm=0,p_time,flag=0;
          system("clear");
  printf("Enter 5 packets in the stream:");
        for(i=0;i<5;++i)
          {
//        packets[i]=t_rand(7)*10;

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

        printf("\nEnter the Output Rate:");
    scanf("%d",&o_rate);
    printf("\nEnter the Bucket Size:");
    scanf("%d",&b_size);
    for(i=0;i<5;++i)
    {
        if((packets[i]+p_sz_rm)>b_size)
        {
            if(packets[i]>b_size)
printf("\n\n Incoming packet size( %d )is GREATER than bucket capacity -
!!!REJECTED!!!",packets[i]);
        else
        printf("\nBucket capacity exceeded - !!!REJECTED!!!");
        }
        else
        {
       for(j=0;;++j)
       {
            p_remain=4-i;
            p_sz=packets[i];
            p_sz_rm+=p_sz;
            printf("\n\n Incoming Packet Size : %d",p_sz);
            printf("\n Transmission Left : %d",p_sz_rm);
            p_time=t_rand(5)*2;
            printf("\n\n Next Packet will come at : %d",p_time);
            for(clk=0;clk<p_time;clk+=1)
            {
                printf("\n Time left : %d",clk);
                sleep(1);
                if(p_sz_rm)
```

```
                    {
                          printf(" - !!!Transmitted!!!");
                          if(p_sz_rm<=o_rate)
                                p_sz_rm=0;
                          else
                                p_sz_rm-=o_rate;
                          printf(" - Bytes Remaining : %d",p_sz_rm);
                    }
                    else
                          printf(" - No Packets to transmit!!!");
              }
              if(p_sz_rm!=0)
                    flag=1;
              break;
         }
          }
     }
  printf("\n\n");
  return(0);
}
```

Output: