```
#include<stdio.h>
#include<conio.h>
#include<iostream>
#include<stdlib.h>
#include<math.h>
#include<dos.h>

using namespace std;
int *Gen;     //*Gen var use for store the value of generate bit
int *Enc;    //*Enc var(pointer) used for store value of encoded bits
class encoding
{
  int K,i,d;    //K=length of genarate bits,
  public:
  void setk(int d)  //setk()function use for set the value of K
  {K=d;}

  void generate1()   //Here the generate function will generate the bits
  {
   for(i=0;i<K;i++)
   Gen[i]=(rand())%2; //this statement is used for generate randomly 1 or 0 bits


   Gen[K]=0;
   Gen[K+1]=0;
   cout<<"Generated bits:-";

   for(i=0;i<K;i++)
   cout<<Gen[i]<<" "; //This statement is used to print generate bits

   i=0;      //i is used to indicated the index num of *Gen
   d=0;     //d is used to indicated the index num of *Enc
  }

//////////////////////////////STATE DIAGRAM/////////////////////////
 int state00()          //here this function representing state00
  {                    //here 00 put in the Enc and again state00() function will call,if the reading bit is
0
   if(Gen[i]==0)
    {
     Enc[d]=0;
     Enc[d+1]=0;
```

```
    d=d+2;
    i++;

    if(i==K+2)
    return 0;

    state00();
   }

  if(Gen[i]==1)    //if the reading bit is 1, 11 bits will put in the Enc and state10() function will call
   {
    Enc[d]=1;
    Enc[d+1]=1;

    d=d+2;
    i++;

    if(i==K+2)
    return 0;

    state10();
   }
 }

int state10()   //as same as above
 {
 if(Gen[i]==0)
  {
   Enc[d]=0;
   Enc[d+1]=1;

   d=d+2;
   i++;

   if(i==K+2)
   return 0;

   state01();
  }

 if(Gen[i]==1)
  {
   Enc[d]=1;
```

```
   Enc[d+1]=0;

   d=d+2;
   i++;

   if(i==K+2)
   return 0;

   state11();
  }
}

int state01()
 {
 if(Gen[i]==0)
  {
   Enc[d]=1;
   Enc[d+1]=1;

   d=d+2;
   i++;

   if(i==K+2)
   return 0;

   state00();
  }

  if(Gen[i]==1)
   {
    Enc[d]=0;
    Enc[d+1]=0;

    d=d+2;
    i++;

    if(i==K+2)
    return 0;

    state10();
   }
 }
```

```
 int state11()
  {
  if(Gen[i]==0)
   {
    Enc[d]=1;
    Enc[d+1]=0;

    d=d+2;
    i++;

    if(i==K+2)
    return 0;

    state01();
   }

  if(Gen[i]==1)
   {
    Enc[d]=0;
    Enc[d+1]=1;

    d=d+2;
    i++;

    if(i==K+2)
    return 0;

    state11();
   }
 }

  print()
  {
   cout<<"\n\nEncoded bits:-";
   for(i=0;i<2*(K+2);i++)    //here i have printed encoded bits
   cout<<Enc[i]<<" ";
   cout<<endl;
  }

};
//////////////////////*!*!*!*!*!*!*!////////////////////////////////////
                        //DCLERATION OF BRANCH MATRIC//
struct stated
```

```
{
                    //here hd[4] is used to store the sum of hamming distance
int hd[4];          //here using the structure ,i can easily create 4*K branch matric
};
////////////////////////*!*!*!*!*!*!/////////////////////////////////


int main()
 {

  int K,channel,i;
  encoding c;               //here c is declare as a class object


  cout<<"How many bits you want to genarate?\n";
  scanf("%d",&K);

   Enc=(int*)malloc(2*(K+2)*sizeof(int)) ;       //dynamic memory allocation for Enc array
   Gen=(int*)malloc((K+2)*sizeof(int)) ;         //dynamic memory allocation for Gen array

  c.setk(K);

  c.generate1();

  c.state00();   //for initial state
  c.print();




  float
SNRdb,SNRdblin,sigma2,sigma,s=1,r,p1[17]={0.1587,0.1447,0.1309,0.1173,0.1040,0.0912,0.07
89,0.0673,0.0565,0.0466,0.0377,0.0298,0.0230,0.0173,0.0126,0.0089,0.0060};
  int Nsim=20000,d,CSNRdb=-1,x2;  //The value of p1 is return by (p=Q()) for 0 to 8 dB;
                          //CSNRdb is used only to denote count1 array's index

  float *recieved1;
  int decode[2*(K+2)],fnldecode[K+2];
  recieved1=(float*)malloc(2*(K+2)*sizeof(float)) ; //dynamic memory allocation for received bits

  int dummy,j,min;
  int Rec[2*(K+2)],d1,d2;
```

```
 struct stated *bm;
 bm=(stated*)malloc((K+5)*sizeof(stated)) ;

 for(int pp=1;pp<=3;pp++) //for 3 different channel
 {
   CSNRdb=-1;
   float count1[17]={0};


  for(SNRdb=0;SNRdb<=8;SNRdb=SNRdb+0.5)
   {  CSNRdb++;
    SNRdblin=pow(10,(SNRdb/10)) ; //pow function will convert dB to lin
    sigma2=1/(2*0.5*SNRdblin);  // rate=0.5
    sigma=sqrt(sigma2);


   for(int x=1;x<=Nsim;x++)  //for 20000 simulation
   {
      if(pp==1)
      {

       for(x2=0;x2<2*(K+2);x2++)     //gaussian channel
        {
         label1:
         r=rand();
         if(r<600+900*CSNRdb||r>10450+CSNRdb*570)  //all of these codes are for random
creation
         goto label1;                 //because in this compiler rand function will generate the value of
the larger digit and positive

         d=r;
         if(d%2==0)
         r=-r;
         r=r/10000;

         if(Enc[x2]==1)
         recieved1[x2]=s+sigma*r;
         else
         recieved1[x2]=-s+sigma*r;

         }
       }
```

```
        if(pp==2)                       //BSC channel
         {
          for(x2=0;x2<2*(K+2);x2++)
          {
           label2:
           r=rand();
           if(r<700-47*CSNRdb||r>50000) //for random creation because here random function will
generate large digit value
           goto label2;

           r=r/10000;

           if(r<p1[CSNRdb])
           recieved1[x2]=(Enc[x2]+1)%2;
           else
           recieved1[x2]=Enc[x2];
          }

         }

        if(pp==3)       //BEC channel
        {
         for(x2=0;x2<2*(K+2);x2++)
          {
           label3:
           r=rand();
           if(r<700-45*CSNRdb||r>50000)
           goto label3;

           r=r/10000;

           if(r<p1[CSNRdb])
           recieved1[x2]=-1;
           else
           recieved1[x2]=Enc[x2];
          }
         }
```

```
////////////////////////////////////////////////////////////////////////
                        //TRELLIS DECODER//
////////////////////////////////////////////////////////////////////////



  int s0outs0[2]={0,0},s0outs1[2]={1,1};     //s0outs0 means state00--00-->state0
  int s1outs2[2]={0,1},s1outs3[2]={1,0};     //s0outs1 means state00--11-->state1
  int s2outs0[2]={1,1},s2outs1[2]={0,0};
  int s3outs3[2]={0,1},s3outs2[2]={1,0};

  for(i=0;i<=K+2;i++)
  {
   bm[i].hd[0]=404;     //this (404) value will never be use but only kept for avoiding random value
of put by compiler
   bm[i].hd[1]=404;
   bm[i].hd[2]=404;
   bm[i].hd[3]=404;
  }
////////////////////////////////////////////////////////////////////////////
 for(i=0;i<2*(K+2);i++)
 {
  if(recieved1[i]<=0)                //We can separate the two from 0
  Rec[i]=0;
  else
  Rec[i]=1;
 }

 ////////////////////////////state0/////////////////////////////////////////
  j=-2;
  bm[0].hd[0]=0;            //Initial value of branch metrice is 0(at 0*0 index)
  for(i=0;i<K+2;i++)
  {
   j=j+2;

    if(bm[i].hd[0]!=404&&bm[i].hd[0]<4)
    {                                    //state0 to state0
     d1=Rec[j]-s0outs0[0];
     d2=Rec[j+1]-s0outs0[1];     //here i am first checking value of Rec(at that time)
     d1=d1* d1;                 //and then count diferent between rec and 00
```

```
    d2=d2*d2;                  //now i will sum d1 and d2 with bm[i].hd[0] and that value set in
bm[i+1].hd[0]
    dummy=bm[i].hd[0]+d1+d2;
    if(bm[i+1].hd[0]!=404&&bm[i+1].hd[0] <dummy) //this condition will set the smallest value in
the branch matric.
    ;
    else
    bm[i+1].hd[0]=dummy;
                                                    //state0 to state1
    d1=Rec[j]-s0outs1[0];
    d2=Rec[j+1]-s0outs1[1]; //here i am first checking value of Rec(at that time)
    d1=d1*d1;
    d2=d2*d2;              //and then count different between rec and 11(because current state is 0)

    dummy=bm[i].hd[0]+d1+d2; ;            //now i will sum d1 and d2 with bm[i].hd[0] and that value
set in bm[i+1].hd[1]
    if(bm[i+1].hd[1]!=404&&bm[i+1].hd[1] <dummy)//this condition will set the smallest value in
the branch matric

    ;
    else
    bm[i+1].hd[1]=dummy;
  }

 ////////////////////////////state1/////////////////////////////////////
                                                      //state1 to state2
  if(bm[i].hd[1]!=404&&bm[i].hd[1]<4)
  {
   d1=Rec[j]-s1outs2[0];
   d2=Rec[j+1]-s1outs2[1];
   d1=d1*d1;                          //as same as above
   d2=d2*d2;

   dummy=bm[i].hd[1]+d1+d2;
   if(bm[i+1].hd[2]!=404&&bm[i+1].hd[2]<dummy)
   ;
   else
   bm[i+1].hd[2]=dummy;

                                                       //state1 to state3
   d1=Rec[j]-s1outs3[0];
   d2=Rec[j+1]-s1outs3[1];
   d1=d1*d1;
```

```
   d2=d2*d2;

   dummy=bm[i].hd[1]+d1+d2;
   if(bm[i+1].hd[3]!=404&&bm[i+1].hd[3]<dummy)
   ;
   else
   bm[i+1].hd[3]=dummy;

  }
//////////////////////state2////////////////////
 if(bm[i].hd[2]!=404&&bm[i].hd[2]<4)
  {                                  //state2 to state1
   d1=Rec[j]-s2outs1[0];
   d2=Rec[j+1]-s2outs1[1];
   d1=d1*d1;
   d2=d2*d2;                          //as same as above

   dummy=bm[i].hd[2]+d1+d2;
   if(bm[i+1].hd[1]!=404&&bm[i+1].hd[1] <dummy)
   ;
   else
   bm[i+1].hd[1]=dummy;                      //state2 to state0

   d1=Rec[j]-s2outs0[0];
   d2=Rec[j+1]-s2outs0[1];
   d1=d1*d1;
   d2=d2*d2;

   dummy=bm[i].hd[2]+d1+d2;
   if(bm[i+1].hd[0]!=404&&bm[i+1].hd[0]<dummy)
   ;
   else
   bm[i+1].hd[0]=dummy;

 }
 ////////////////////////state3/////////////////////////
if(bm[i].hd[3]!=404&&bm[i].hd[3]<4)
 {                                  //state3 to state3
  d1=Rec[j]-s3outs3[0];
  d2=Rec[j+1]-s3outs3[1];
  d1=d1*d1;
  d2=d2*d2;
```

```
   dummy=bm[i].hd[3]+d1+d2;
   if(bm[i+1].hd[3]!=404&&bm[i+1].hd[3]<dummy)
   ;
   else
   bm[i+1].hd[3]=dummy;

   d1=Rec[j]-s3outs2[0];
   d2=Rec[j+1]-s3outs2[1];                    //state3 to state2
   d1=d1*d1;
   d2=d2*d2;

   dummy=bm[i].hd[3]+d1+d2;
   if(bm[i+1].hd[2]!=404&&bm[i+1].hd[2]<dummy)
    ;
    else
    bm[i+1].hd[2]=dummy;

   }
 }
////////////////////////////////////////////////////////////////////////
                              //the process of finding correct path and decoding //
////////////////////////////////////////////////////////////////////////
   //here decode var will store the convolution decode and fnldecode will store the actual bits
sent
 int  cs=0,set;
 j=2*(K+2)-1;

  for(i=K+2;i>0;i--)
  {
   ////////////current state 0//////////////////
   if(cs==0)                      //here cs representing current state
    {
      set=0;
      d1=0-Rec[j];  //here the whole process has started from the last of path matric
      d2=0-Rec[j-1];  //Here I have the same process as branch metric.
      d1=d1*d1;
      d2=d2*d2;

      min=bm[i-1].hd[0]+d1+d2; //Because I want to check which of previous two branch
matric(depend on state) is in bm[i].hd[0]
      if(min==bm[i].hd[0])  ///1///
       {
```

```
            decode[j]=0;
            decode[j-1]=0;
            cs=0;
            set=1;
            fnldecode[i-1]=0;
     }
                    //one of the two condition which is correct will be stored(Path) in Decode array
and store the corresponding(actual generated bit) bit in the fnldecode array (since last)
    d1=1-Rec[j];
          d2=1-Rec[j-1];
          d1=d1*d1;
          d2=d2*d2;
          min=bm[i-1].hd[2]+d1+d2;
   if(set!=1||(d1+d2)==0) ///2///
          {
           decode[j]=1;
           decode[j-1]=1;
     cs=2;
           fnldecode[i-1]=0;
          }
   }
//////////////////////current state 1//////////////////////
    else if(cs==1)
     {
      set=0;
      d1=1-Rec[j];
      d2=1-Rec[j-1];

      d1=d1*d1;
      d2=d2*d2;

      min= d2+d1+bm[i-1].hd[0];

      if(min==bm[i].hd[1])
            {                          //as same as above(only branches are diffrent
             decode[j]=1;
             decode[j-1]=1;
             cs=0;
             set=1;
              fnldecode[i-1]=1;
            }

          d1=0-Rec[j];
```

```
            d2=0-Rec[j-1];
            d1=d1*d1;
            d2=d2*d2;
            min=bm[i-1].hd[2]+d1+d2;
           if(set!=1||(d1+d2)==0)
           {
            decode[j]=0;
            decode[j-1]=0;
            cs=2;
            fnldecode[i-1]=1;
           }
      }
////////////////////current state 2/////////////////////////////////////
      else if(cs==2)
      {
       set=0;
       d1=1-Rec[j];
       d2=0-Rec[j-1];
       d1=d1*d1;
       d2=d2*d2;
       min=bm[i-1].hd[1]+d1+d2;
      if(min==bm[i].hd[2])
       {
              decode[j]=1;
              decode[j-1]=0;
        cs=1;
              set=1;
              fnldecode[i-1]=0;
           }
             d1=0-Rec[j];
             d2=1-Rec[j-1];
      d1=d1*d1;
      d2=d2*d2;
      min=bm[i-1].hd[3]+d1+d2;
    if(set!=1||(d1+d2)==0)
           {
            decode[j]=0;
            decode[j-1]=1;
            cs=3;
            fnldecode[i-1]=0;
           }
    }
////////////////////////////////////current state 3/////////////////////////////////////
```

```
     else if(cs==3)
      {
       set=0;
       d1=0-Rec[j];
       d2=1-Rec[j-1];

       d1=d1*d1;
       d2=d2*d2;
       min=bm[i-1].hd[1]+d1+d2;
      if(bm[i].hd[3]==min)
       {
              decode[j]=0;
        decode[j-1]=1;
        cs=1;
              set=1;
              fnldecode[i-1]=1;
       }
       d1=1-Rec[j];
       d2=0-Rec[j-1];
       d1=d1*d1;
       d2=d2*d2;
       min=bm[i-1].hd[3]+d1+d2;
      if(set!=1||(d1+d2)==0)
             {
              decode[j]=1;
              decode[j-1]=0;
        cs=3;
              fnldecode[i-1]=1;
             }
      }
      j=j-2;
    }

//bits error count//
  for(i=0;i<K+2;i++)
 count1[CSNRdb]=(count1[CSNRdb]+(fnldecode[i]+Gen[i])%2);// fnldecode contain decoding
bits and Gen contain generated bits
    //if the decoding and generated bits are different then 1 will be added to count1[CSNRdb]
 }
}

 if(pp==1)
    cout<<"Gaussian channel:- ";
```

```
 else if(pp==2)
    cout<<"BSC channel:- ";
 else
    cout<<"BEC Channel:- ";

  for(i=0;i<CSNRdb;i++)
    {
     count1[i]=count1[i]/((K+2)*(20000));  //here K+2=500
     cout<<count1[i]<<" ";
    }cout<<endl;

  }
getch();
return 0;
}
```