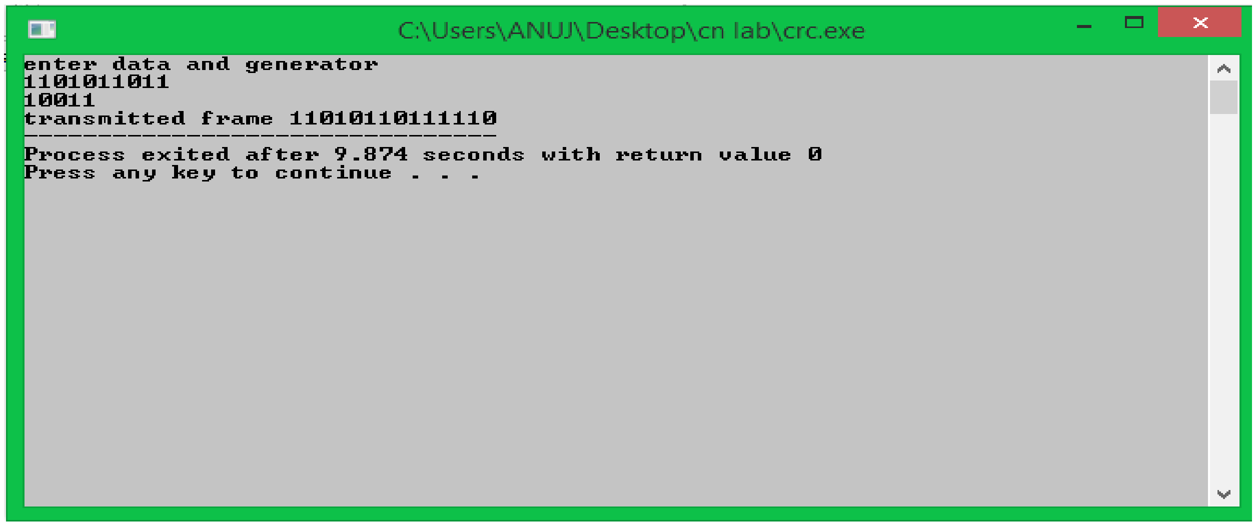
code for CRC

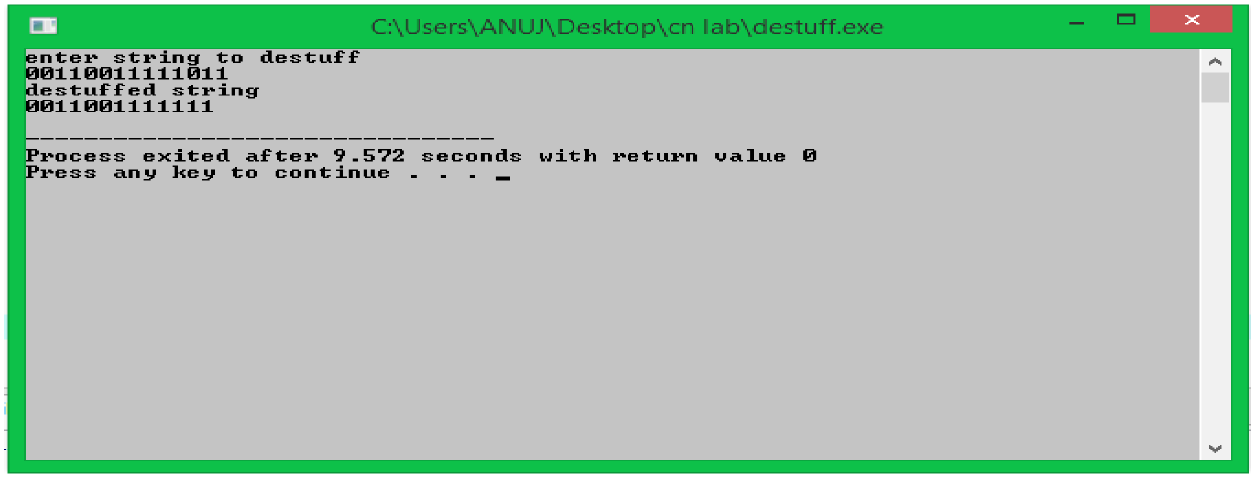
```
#include<iostream>
using namespace std;
char xor1(char a,char b)
{
        if(a!=b)
         return '1';
        else
        return '0';
}
int main()
{
        string divsr,divdnd,quot,rem;
        cout<<"enter data and generator";
        cin>>divdnd;
        cin>>divsr;
        int t=divdnd.length();
        for(int m=0;m<divsr.length()-1;m++)
        {
          divdnd =divdnd+'0';
        }
        string h=divdnd;
        int k=0,l=0;
 for(int j=0;j<=(divdnd.length()-divsr.length());j++)
 {
        if(divdnd[j]=='1')
        {  quot[k++]='1';
           for(int i=0;i<divsr.length();i++)


                 {
                   divdnd[i+j]=xor1(divdnd[i+j],divsr[i]);
           }
        }
        else
        {
                quot[k++]='0';
        }
}
int p=0,m=divdnd.length()-divsr.length()+1;
for(int i=(divdnd.length()-divsr.length()+1);i<=divdnd.length();i++)
{
        cout<<divdnd[i]<<endl;
        h[m++]= xor1(divdnd[i],'0' );
}
cout<<"transmitted frame"<<h;
}
```



```
enter data and generator
1101011011
10011
transmitted frame 11010110111110
--------------------------------
Process exited after 9.874 seconds with return value 0
Press any key to continue . . .
```

<u>code for bit destuffing</u>

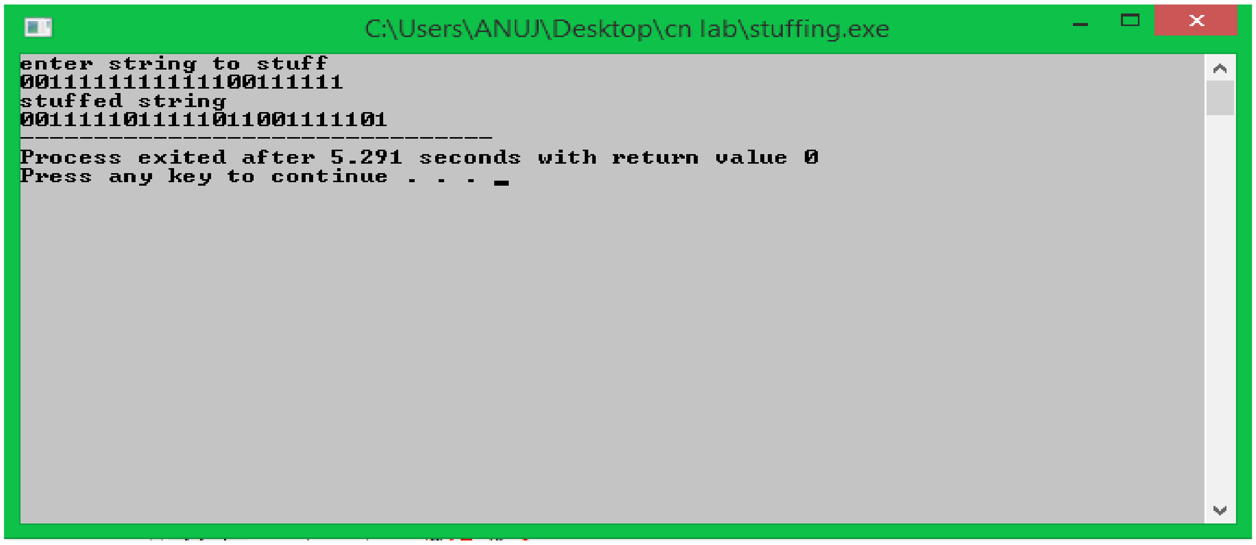
```
#include<iostream>
using namespace std;
int main()
{
        string s,m="";
        cout<<"enter string to destuff"<<endl;
        cin>>s;
        int count=0,k=0;
        for(int i=0;i<s.length();i++)
        {
          if(s[i]=='0')
          {
          count=0;m+=s[i];
   }
                if(s[i]=='1')
                 {
                count++;m+=s[i];
        }
                if(count==5)
                 {
                        count=0;
                        i=i+2;
                        m+=s[i];
         }
  }
        cout<<"destuffed string"<<m<<endl;
}
```



```
enter string to destuff
00110011111011
destuffed string
0011001111111

--------------------------------
Process exited after 9.572 seconds with return value 0
Press any key to continue . . . _
```

//code for bit stuffing

```
#include<iostream>
using namespace std;
int main()
{
            string s,m="";
            cout<<"enter string to stuff";
            cin>>s;
            int count=0,k=0;
            for(int i=0;i<s.length();i++)
            {
              if(s[i]=='0')
              {
              count=0;m+=s[i];
     }
                        if(s[i]=='1')
                         {
                        count++;m+=s[i];
            }
                        if(count==5)
                         {
                                    count=0;
                                    m+='0';
             }
   }
            cout<<"stuffed string"<<m;
}
```



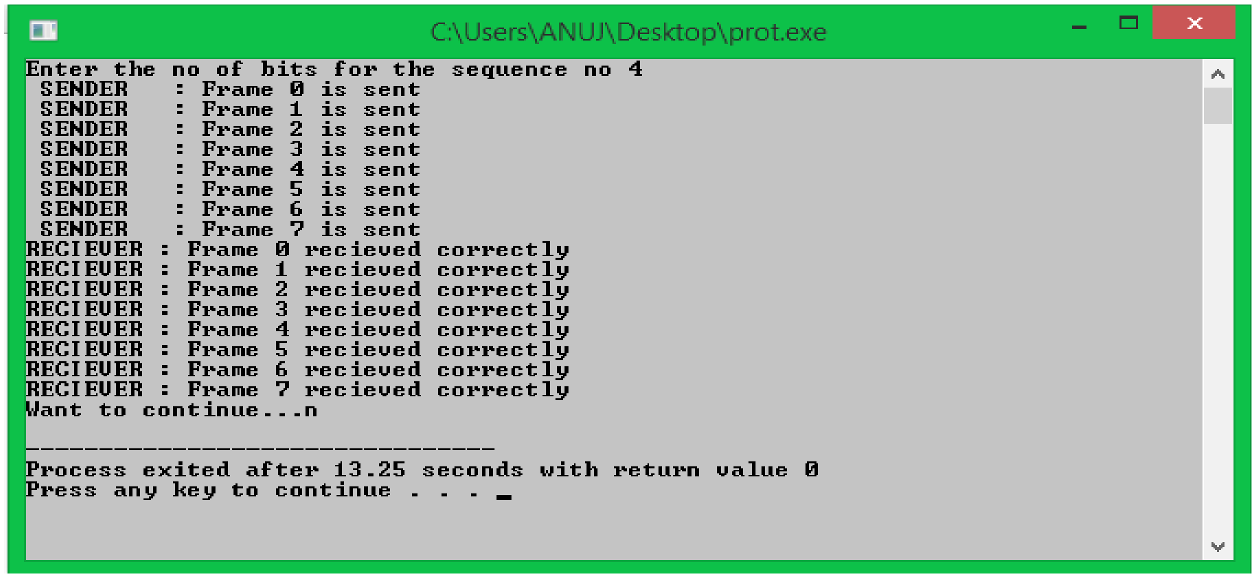
```
enter string to stuff
0011111111111100111111
stuffed string
0011111011111011001111101
--------------------------------
Process exited after 5.291 seconds with return value 0
Press any key to continue . . .
```

go back n protocol

```
# include <iostream>
# include <conio.h>
# include <stdlib.h>
# include <time.h>
# include <math.h>
# define TOT_FRAMES 500
# define FRAMES_SEND 10
using namespace std;
class gobkn
{
 private:
  int fr_send_at_instance;
  int arr[TOT_FRAMES];
  int arr1[FRAMES_SEND];
  int sw;
  int rw; // tells expected frame
 public:
  gobkn();
  void input();
  void sender(int);
  void reciever(int);
};

gobkn :: gobkn()
{
 sw = 0;
 rw = 0;
}

void gobkn :: input()
{
 int n;  // no of bits for the frame
 int m;  // no of frames from n bits

 cout << "Enter the no of bits for the sequence no ";
 cin >> n;
 m = pow (2 , n);

 int t = 0;

 fr_send_at_instance = (m / 2);

 for (int i = 0 ; i < TOT_FRAMES ; i++)
 {
  arr[i] = t;
  t = (t + 1) % m;
 }
 sender(m);
}
void gobkn :: sender(int m)
{
 int j = 0;
 for (int i = sw ; i < sw + fr_send_at_instance ; i++)
 {
  arr1[j] = arr[i];
  j++;
 }
 for (int i = 0 ; i < j ; i++)
  cout << " SENDER   : Frame " << arr1[i] << " is sent\n";
 reciever (m);
}
void gobkn :: reciever(int m)
{
 time_t t;
 int f;
 int f1;
 int a1;
 char ch;
 srand((unsigned) time(&t));
 f = rand() % 10;
   // if = 5 frame is discarded for some reason
   // else they are correctly recieved
 if (f != 5)
 {
```

```
 for (int i = 0 ; i < fr_send_at_instance ; i++)
 {
  if (rw == arr1[i])
  {
   cout << "RECIEVER : Frame " << arr1[i] << " recieved correctly\n";
   rw = (rw + 1) % m;
  }
  else
   cout << "RECIEVER : Duplicate frame " << arr1[i] << " discarded\n";
 }
 a1 = rand() % 15;
  // if a1 belongs to 0 to 3 then
  //     all ack after this (incl this one) lost
  // else
  //     all recieved
 if (a1 >= 0 && a1 <= 3)
 {
  cout << "(Acknowledgement " << arr1[a1] << " & all after this lost)\n";
  sw = arr1[a1];
 }
 else
  sw = (sw + fr_send_at_instance) % m;
}
else
{
 f1 = rand() % fr_send_at_instance;
// f1 gives index of the frame being lost
 for (int i = 0 ; i < f1 ; i++)
 {
  if (rw == arr1[i])
  {
   cout << " RECIEVER : Frame " << arr1[i] << " recieved correctly\n";
   rw = (rw + 1) % m;
  }
  else
   cout << " RECIEVER : Duplicate frame " << arr1[i] << " discarded\n";
 }
 int ld = rand() % 2;
```

```
   // ld == 0 frame damaged
   // else frame lost
  if (ld == 0)
   cout << " RECIEVER : Frame " << arr1[f1] << " damaged\n";
  else
   cout << "           (Frame " << arr1[f1] << " lost)\n";

  for (int i = f1 + 1 ; i < fr_send_at_instance ; i++)
    cout << " RECIEVER : Frame " << arr1[i] << " discarded\n";

  cout << " (SENDER TIMEOUTS --> RESEND THE FRAME)\n";

  sw = arr1[f1];
 }
 cout << "Want to continue...";
 cin >> ch;
 if (ch == 'y')
  sender(m);
 else
  exit(0);
}
int  main()
{
 gobkn gb;
 gb.input();
 getch();
}
```



C:\Users\ANUJ\Desktop\prot.exe

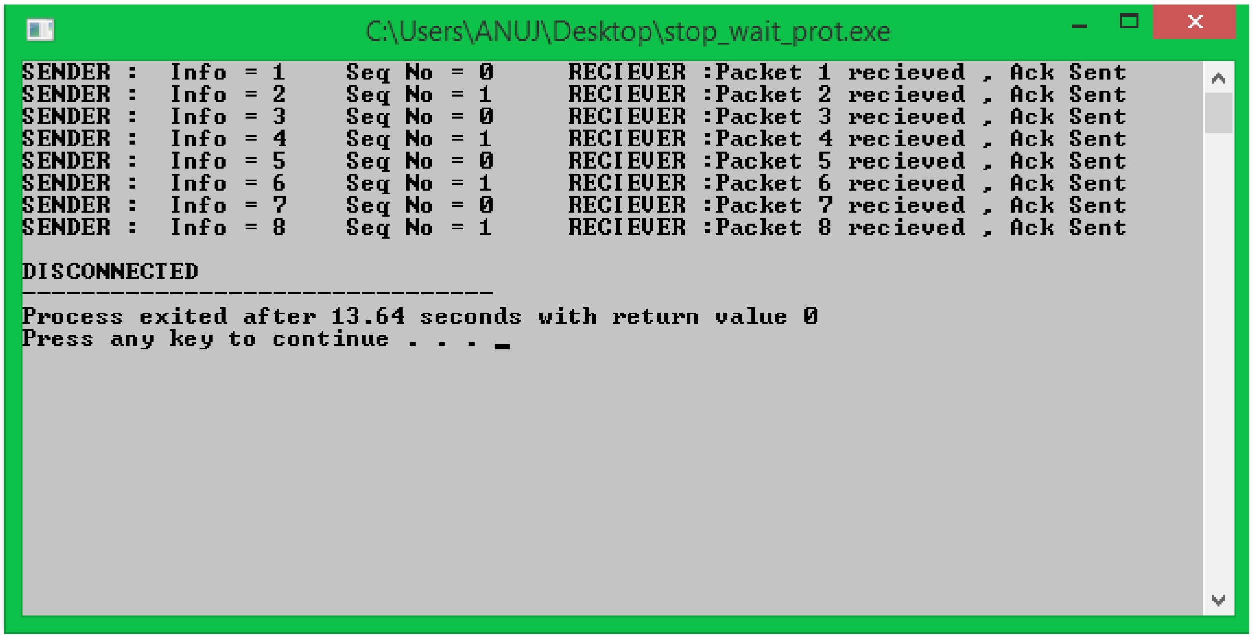
```
Enter the no of bits for the sequence no 4
 SENDER    : Frame 0 is sent
 SENDER    : Frame 1 is sent
 SENDER    : Frame 2 is sent
 SENDER    : Frame 3 is sent
 SENDER    : Frame 4 is sent
 SENDER    : Frame 5 is sent
 SENDER    : Frame 6 is sent
 SENDER    : Frame 7 is sent
RECIEVER : Frame 0 recieved correctly
RECIEVER : Frame 1 recieved correctly
RECIEVER : Frame 2 recieved correctly
RECIEVER : Frame 3 recieved correctly
RECIEVER : Frame 4 recieved correctly
RECIEVER : Frame 5 recieved correctly
RECIEVER : Frame 6 recieved correctly
RECIEVER : Frame 7 recieved correctly
Want to continue...n

--------------------------------
Process exited after 13.25 seconds with return value 0
Press any key to continue . . . _
```

program for wait and stop

```
#include <conio.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <bits/stdc++.h>
#define TIMEOUT 5
#define MAX_SEQ 1
#define TOT_PACKETS 8
#define inc(k) if(k<MAX_SEQ) k++; else k=0;
using namespace std;
typedef struct
{
 int data;
}packet;
typedef struct
{
 int kind;
 int seq;
 int ack;
 packet info;
 int err;
}frame;
frame DATA;
typedef enum{frame_arrival,err,timeout,no_event} event_type;

void from_network_layer(packet *);
void to_network_layer(packet *);
void to_physical_layer(frame *);
void from_physical_layer(frame *);
void wait_for_event_sender(event_type *);
void wait_for_event_reciever(event_type *);
void reciever();
void sender();

int i=1;        //Data to be sent by sender
char turn;     //r , s
```

```
int DISCONNECT=0;
/*__________________________________________________________________________*/
int main()
{

 rand();
 while(!DISCONNECT)
 {
   sender();
  sleep(1);
   reciever();
 }
 getch();
}
/*__________________________________________________________________________*/
void sender()
{
 static int frame_to_send=0;
 static frame s;
 packet buffer;
 event_type event;
 static int flag=0;

 if(flag==0)
 {
  from_network_layer(&buffer);
  s.info = buffer;
  s.seq = frame_to_send;
  printf("SENDER :  Info = %d    Seq No = %d     ",s.info,s.seq);
  turn = 'r';
  to_physical_layer(&s);
  flag = 1;
 }
 wait_for_event_sender(&event);
 if(turn=='s')
  {
   if(event==frame_arrival)
    {
     from_network_layer(&buffer);
     inc(frame_to_send);
     s.info = buffer;
     s.seq = frame_to_send;
     printf("SENDER :  Info = %d    Seq No = %d     ",s.info,s.seq);
     turn = 'r';
     to_physical_layer(&s);
    }
   if(event==timeout)
    {
     printf("SENDER : Resending Frame             ");
     turn = 'r';
     to_physical_layer(&s);
    }
  }
}
/*________________________________________________________________________*/
void reciever()
{
 static int frame_expected=0;
 frame r,s;
 event_type event;

 wait_for_event_reciever(&event);
 if(turn=='r')
  {
   if(event==frame_arrival)
    {
    from_physical_layer(&r);
    if(r.seq==frame_expected)
     {
     to_network_layer(&r.info);
     inc(frame_expected);
     }
    else
     printf("RECIEVER : Acknowledgement Resent\n");

    turn = 's';
    to_physical_layer(&s);
   }
  if(event==err)
   {
     printf("RECIEVER : Garbled Frame\n");
     turn = 's';     //if frame not recieved
   }                //sender shold send it again
 }
}
/*___________________________________________________________________________*/
void from_network_layer(packet *buffer)
{
  (*buffer).data = i;
  i++;
}
/*____________________________________________________________________________*/
void to_physical_layer(frame *s)
{                   // 0 means error
 s->err = rand();  //non zero means no error
 DATA = *s;          //probability of error = 1/4
}
/*____________________________________________________________________________*/
void to_network_layer(packet *buffer)
{
 printf("RECIEVER :Packet %d recieved , Ack Sent\n",(*buffer).data);
 if(i>TOT_PACKETS)         //if all packets recieved then disconnect
  {
   DISCONNECT = 1;
   printf("\nDISCONNECTED");
  }
}
/*____________________________________________________________________________*/
void from_physical_layer(frame *buffer)
{
 *buffer = DATA;
}
/*____________________________________________________________________________*/
void wait_for_event_sender(event_type * e)
```

```
{
 static int timer=0;

 if(turn=='s')
  {
   timer++;
   if(timer==TIMEOUT)
    {
     *e = timeout;
     printf("SENDER : Ack not recieved=> TIMEOUT\n");
     timer = 0;
     return;
    }
   if(DATA.err==0)
     *e = err;
   else
    {
     timer = 0;
     *e = frame_arrival;
    }
  }
}
/*___________________________________________________________________________*/
void wait_for_event_reciever(event_type * e)
{
 if(turn=='r')
  {
   if(DATA.err==0)
     *e = err;
   else
     *e = frame_arrival;
  }
}
```



```
SENDER :  Info = 1     Seq No = 0      RECIEVER :Packet 1 recieved , Ack Sent
SENDER :  Info = 2     Seq No = 1      RECIEVER :Packet 2 recieved , Ack Sent
SENDER :  Info = 3     Seq No = 0      RECIEVER :Packet 3 recieved , Ack Sent
SENDER :  Info = 4     Seq No = 1      RECIEVER :Packet 4 recieved , Ack Sent
SENDER :  Info = 5     Seq No = 0      RECIEVER :Packet 5 recieved , Ack Sent
SENDER :  Info = 6     Seq No = 1      RECIEVER :Packet 6 recieved , Ack Sent
SENDER :  Info = 7     Seq No = 0      RECIEVER :Packet 7 recieved , Ack Sent
SENDER :  Info = 8     Seq No = 1      RECIEVER :Packet 8 recieved , Ack Sent

DISCONNECTED
--------------------------------
Process exited after 13.64 seconds with return value 0
Press any key to continue . . .
```