INDRAPRASTHA COLLEGE FOR WOMEN

(31, Shamnath Marg, Civil Lines, New Delhi-110054)

(University of Delhi)

DATA COMMUNICATION AND COMPUTER NETWORKS PRACTICAL

MADE BY-
Eva Chaudhary
IV SEMESTER

INDEX

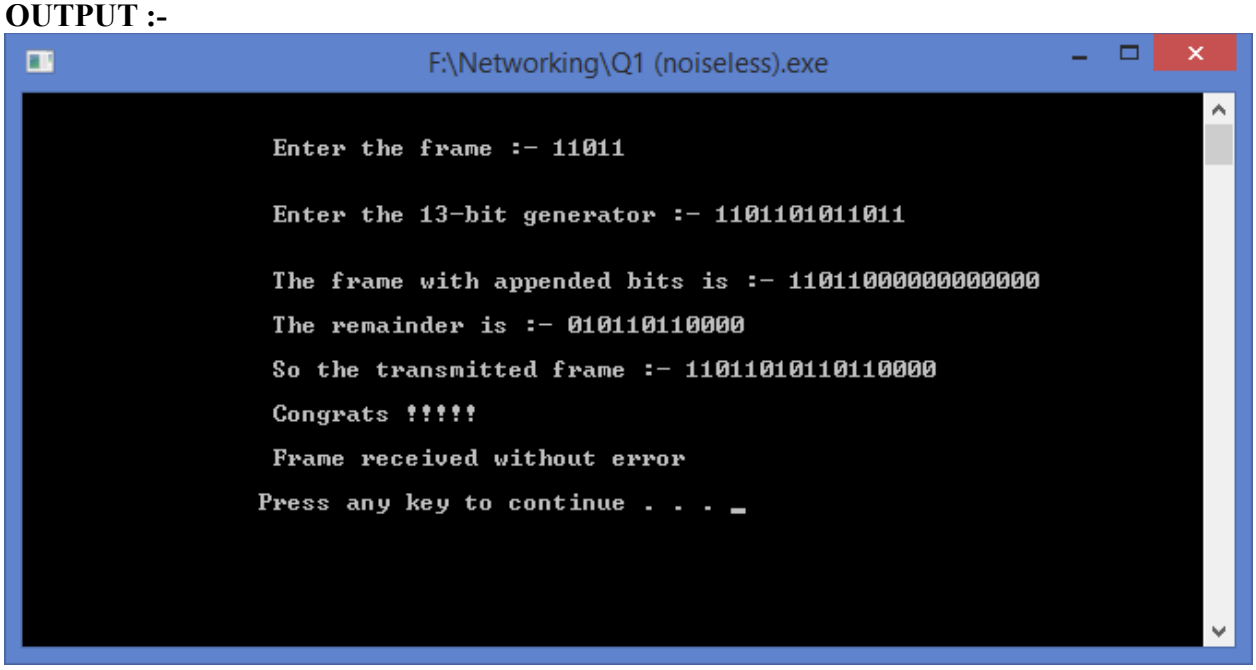
Ques 1) Implement CRC-12 for Noiseless channel.
:-

```
using namespace std;
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<ctime>
#include<cstring>

#define maxg 13
#define maxf 30

void remder(char *f,char *g,char *r)
{
    char *tf,*tg,*dev,*div;
    int lenf,leng,temp;
    lenf=strlen(f);
    leng=strlen(g);
    tf=new char[lenf];


    tg=new char[leng];
    dev=new char[leng];
    div=new char[leng];
    strcpy(tf,f);
    strcpy(tg,g);
    for(int i=0;i<leng;i++)
    {
        dev[i]=tf[i];
    }
    int count=leng;
    while(count<=lenf)
    {
            if(dev[0]=='1')
            {
                strcpy(div,tg);
                for(int j=leng-1;j>=0;j--)
                {
                    if(dev[j]==div[j])
                    dev[j]='0';
                    else
                    dev[j]='1';
                }
            }
            for(int i=0;i<leng-1;i++)
            {
```

```
                        temp=dev[i+1];
                        dev[i]=temp;
                    }
                    dev[leng-1]=tf[count];
                    count++;
    }
    for(int k=0;k<leng-1;k++)
    {
        r[k]=dev[k];
    }
}

void check_frame(char *f,char *g)
{
    int flag;
    char *r;
    r=new char[maxg-1];
    //strcpy(r,"000000000000");
    remder(f,g,r);
    for(int i=0;i<maxg-1;i++)
    {
        if(r[i]=='0')
        {
                flag=0;
        }
        else
        {
            flag=1;
            break;
        }
    }
    if(flag==1)
    {
                cout<<"\n\n\t\t Sorry !!!!!  .";
                cout<<"\n\n\t\t Frame received with error \n ";
    }
        else
    {
                cout<<"\n\n\t\t Congrats !!!!! ";
                cout<<"\n\n\t\t Frame received without error ";
    }
}


int main()
{
```

```
        char *frame,*gen,*rem,*f,*d;
        frame=new char[maxf];
        gen=new char[maxg];
    rem=new char[maxg-1];
    d=new char[maxf];
    f=new char[maxf];
        cout<<"\n\n\t\t Enter the frame :- ";
        cin>>frame;
        strcpy(rem,"000000000000");
        cout<<"\n\n\t\t Enter the 13-bit generator :- ";
        cin>>gen;
        strcpy(f,frame);
        strcat(f,rem);
        cout<<"\n\n\t\t The frame with appended bits is :- ";
        cout<<f;
        remder(f,gen,rem);
        cout<<"\n\n\t\t The remainder is :- "<<rem;
        strcpy(d,frame);
        strcat(d,rem);
        cout<<"\n\n\t\t So the transmitted frame :- "<<d;
        check_frame(d,gen);
        cout<<"\n\n\t\t";
        system("pause");
        return 0;
}
```

OUTPUT :-



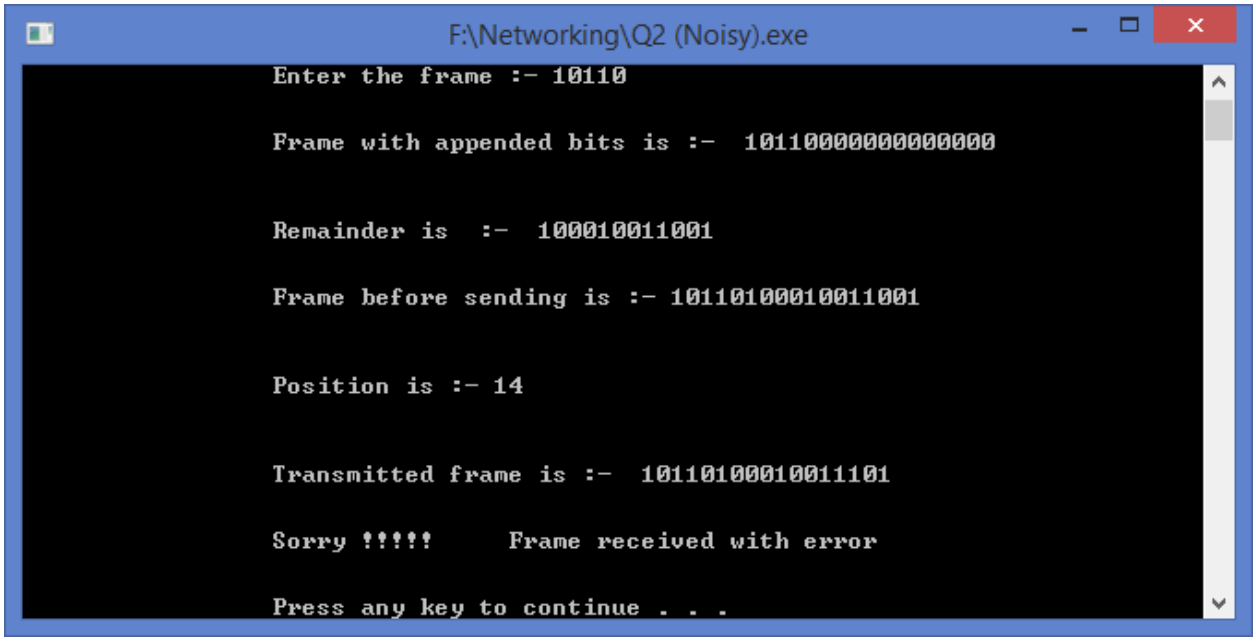
Ques 2) Implement CRC-12 with Noisy Channel.
:-

```
using namespace std;
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<ctime>
#include<cstring>
#define maxf 40
#define maxg 13

void remder(char *f,char *g,char *r)
{
    char *tf,*tg,*dev,*div;
    int lenf,leng,temp;
    lenf=strlen(f);
    leng=strlen(g);
    tf=new char[lenf];
    tg=new char[leng];
    dev=new char[leng];
    div=new char[leng];
    strcpy(tf,f);
    strcpy(tg,g);
    for(int i=0;i<leng;i++)
    {
        dev[i]=tf[i];
    }
    int count=leng;
    while(count<=lenf)
    {
            if(dev[0]=='1')
            {
                strcpy(div,tg);
                for(int j=leng-1;j>=0;j--)
                {
                    if(dev[j]==div[j])
                    dev[j]='0';
                    else
                    dev[j]='1';
                }
            }
            for(int i=0;i<leng-1;i++)
            {
                temp=dev[i+1];
                dev[i]=temp;
            }
```

```
                dev[leng-1]=tf[count];
                count++;
        }
        for(int k=0;k<leng-1;k++)
        {
            r[k]=dev[k];
        }
    }
    void transmt_frame(char *s,char *f)
    {
        char e='1';
        int len=strlen(s);
        srand(time(NULL));
        int pos=rand()%len+1;
        cout<<"\n\n\t\t Position is :- "<<pos<<"\n\n";
        strcpy(f,s);
        f[pos]=e;
    }
    void check_frame(char *f,char *g)
    {
        int flag;
        char *r;
        r=new char[maxg-1];
        strcpy(r,"000000000000");
        remder(f,g,r);
        for(int i=0;i<maxg-1;i++)
        {
            if(r[i]=='0')
            {
                flag=0;
            }
            else
            {
                flag=1;
                break;
            }
        }
        if(flag==1)
          cout<<"\n\n\t\t Sorry !!!!! \t Frame received with error \n ";
        else
          cout<<"\n\n\t\t Congrats !!!!! \t Frame received without error ";
    }

    int main()
    {
       system("cls");
```

```
    char *frame,*gen,*rem,*s,*d,*f;
    frame=new char[maxf];
    gen=new char[maxg];
    rem=new char[maxg-1];
    s=new char[maxf];
    d=new char[maxf];
    f=new char[maxf];
    strcpy(gen,"1100000001111");
    strcpy(rem,"000000000000");
    cout<<"\n\n\t\t Enter the frame :- ";
    cin>>frame;
    strcpy(s,frame);
    strcat(s,rem);
    cout<<"\n\n\t\t Frame with appended bits is :-  "<<s<<"\n\n";
    remder(s,gen,rem);
    cout<<"\n\n\t\t Remainder is  :-  "<<rem<<"\n";
    strcpy(d,frame);
    strcat(d,rem);
    cout<<"\n\n\t\t Frame before sending is :- "<<d<<"\n\n";
    transmt_frame(d,f);
    cout<<"\n\n\t\t Transmitted frame is :-  "<<f<<"\n";
    check_frame(f,gen);
    cout<<"\n\n\t\t ";
    system("pause");
    return 0;
}
```

OUTPUT :-



Ques 3) Implement Stop and Wait Protocol for Noiseless channel.
:-

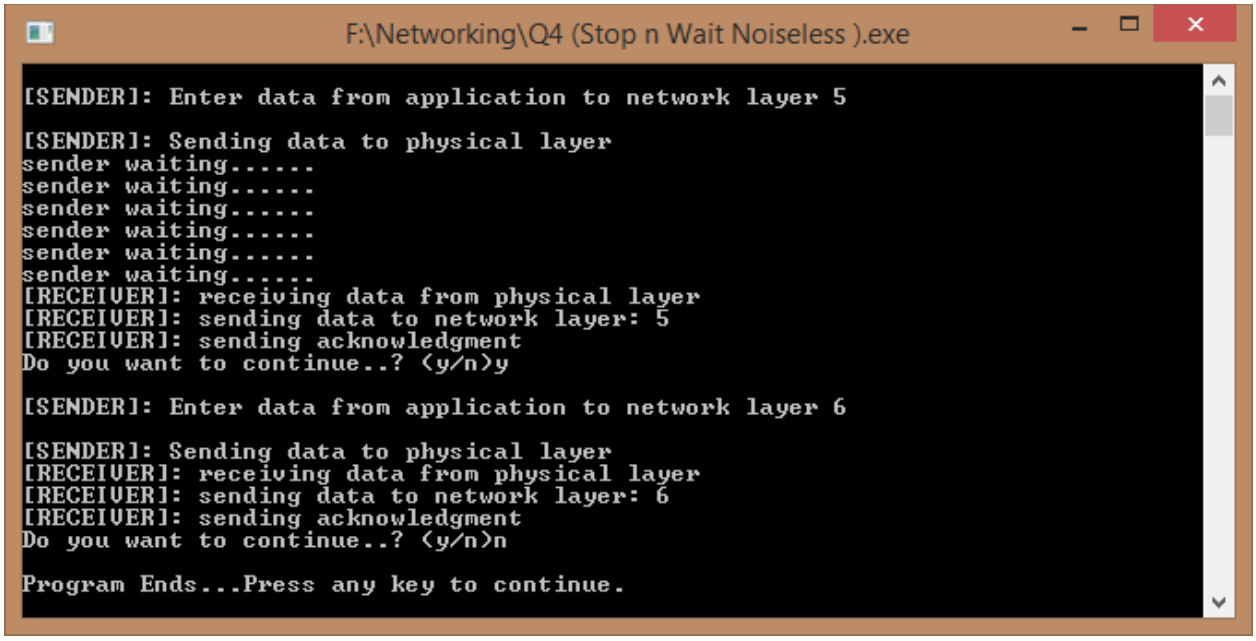
```
#include<iostream>
#include<windows.h>
#include<conio.h>
using  namespace std;

typedef enum {ACK,DAT} frame_kind;

typedef enum { ready_to_send, frame_arrival,frame_send,waiting} event_type;

event_type current_event;

class packet
{
        public: char data[100];
};
class frame
{
        public:
        packet info;
        frame_kind kind;
};

frame common_medium;
class station
{
        public:
        packet from_network_layer()
  {
        //create a new packet and return it
          packet p;
    cout<<"\n[SENDER]: Enter data from application to network layer ";
    cin>>p.data;
    return p;
  }

  void  to_physical_layer(frame s)
 {
        //transfer the frame to physical layer
   cout<<"\n[SENDER]: Sending data to physical layer ";
   common_medium=s;
 }

 event_type wait_for_event()
 {
```

```
        if(common_medium.kind == ACK)
                return ready_to_send;

        if(common_medium.kind == DAT)
                return frame_arrival;

        return ready_to_send;
}

void send()
{
        current_event = wait_for_event();
        if(current_event == ready_to_send)
        {
                //generate a packet from network layer
                  packet p = from_network_layer();

                //convert packet into frame
                frame s;
                s.info = p;
                s.kind = DAT;

                //pass onto physical layer
                to_physical_layer(s);

                //set event frame arrival so that receiver can receieve the packet
                current_event = frame_arrival;
  }
  else
  {
        cout<<"\nsender waiting.....";
        }
}
        void to_network_layer(packet p)
 {
   cout<<"\n[RECEIVER]: sending data to network layer: ";
   cout<<p.data;

 }
frame from_physical_layer()
{
        frame f;
  cout<<"\n[RECEIVER]: receiving data from physical layer ";
  f=common_medium;
  return f;
 }
```

```
  void receive()
  {
        current_event = wait_for_event();
        if(current_event == frame_arrival)
        {
                //read a frame via common medium
                frame s = from_physical_layer();

                //convert the frame into a packet
                packet p;
                p = s.info;

                //send the packet to network layer
                to_network_layer(p);


                //send acknowledgement via common medium
                frame ack;
                ack.kind = ACK;
                cout<<"\n[RECEIVER]: sending acknowledgment";
                common_medium = ack;

        }
        else
   {
        cout<<"\nReceiver waiting.....";
        }
  }
};

int main()
{
        station S;
        char choice = 'y';
        while(choice =='y' || choice == 'Y')
        {
          S.send();
          Sleep(1000);

          int r = rand()%3;
          if(r==0)
          {
           S.receive();
           cout<<"\nDo you want to continue..? (y/n)";
           cin>>choice;
```

```
            }

    }
    cout<<"\nProgram Ends...Press any key to continue.";

    getch();

            return 0;
}
```

OUTPUT:



Ques 4) Implement Stop and Wait for Noisy Channel.

:-

```
#include<iostream>
#include<windows.h>
#include<conio.h>

#define MAX_TIME_OUT 5
using  namespace std;

typedef enum {ACK,DAT} frame_kind;

typedef enum { ready_to_send, frame_arrival,frame_send,waiting,time_out,frame_resend}
event_type;

class packet
{
        public: char data[100];
};
```

```
class frame
{
        public:
        int sno;
        packet info;
        frame_kind kind;
};

frame common_medium;
event_type current_event;
int next_seq_no = 0;

class station
{
        public:
   int timer;
   frame last_frame;

        packet from_network_layer()
   {
          //create a new packet and return it
          packet p;
    cout<<"\n[SENDER]: Enter data to be sent ";
    cin>>p.data;
    return p;
   }

   void  to_physical_layer(frame s)
  {
        //transfer the frame to physical layer
    cout<<"\n[SENDER]: sending frame to physical layer  (sno="<<s.sno<<")";
    common_medium=s;
  }

  event_type wait_for_event()
  {
   if(timer == MAX_TIME_OUT)
      return time_out;

   if(common_medium.kind==ACK && common_medium.sno != next_seq_no)
      return frame_resend;

        if(common_medium.kind == ACK)
                return ready_to_send;

        if(common_medium.kind == DAT)
```

```
                return frame_arrival;

        return ready_to_send;
}

void send()
{
        current_event = wait_for_event();

        if(current_event == ready_to_send)
        {
                //generate a packet from network layer
           packet p = from_network_layer();

                //convert packet into frame
                frame s;
                s.sno = next_seq_no;
                s.info = p;
                s.kind = DAT;

                //store this frame as last sent frame
                last_frame = s;

                //update the next sequence number
                next_seq_no = (next_seq_no+1)%2;

                //pass onto physical layer
                to_physical_layer(s);

                //set event frame arrival so that receiver can receieve the packet
                current_event = frame_arrival;

                //reset the timer;
                timer = 0;
  }
  else if(current_event == time_out || current_event == frame_resend)
  {
     cout<<"\n[SENDER]: Repeat frame sending due to "<<show(current_event);

     //resend the previous frame due to time out
     to_physical_layer(last_frame);

                //set event frame arrival so that receiver can receieve the packet
                current_event = frame_arrival;

                //reset the timer;
```

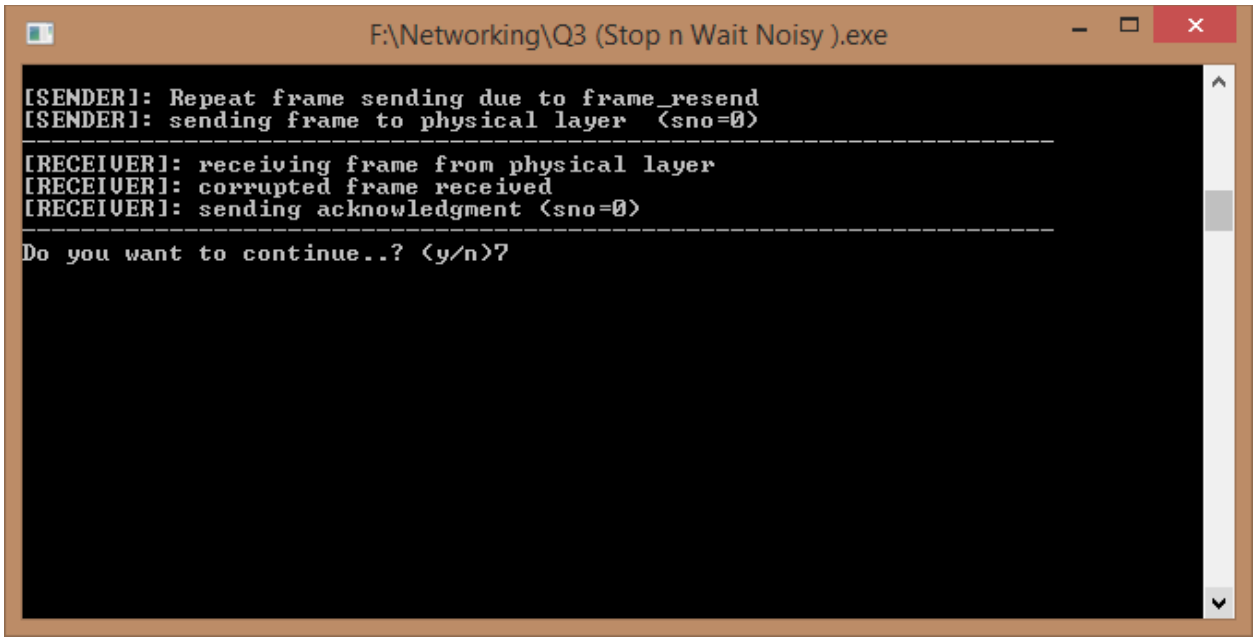
```
                timer = 0;
    }
    else
    {
        cout<<"\nsender waiting for acknowledgment.....";
        timer++;
        cout<<"timer = "<<timer;
        }

        cout<<"\n-----------------------------------------------------------------";
  }
        void to_network_layer(packet p)
   {
     cout<<"\n[RECEIVER]: sending packet to network layer: ";
     cout<<p.data;

   }
  frame from_physical_layer()
  {
        frame f;
    cout<<"\n[RECEIVER]: receiving frame from physical layer ";
    f=common_medium;
    return f;
   }

   void receive()
   {
        current_event = wait_for_event();
        if(current_event == frame_arrival)
        {
      frame ack;

                //read a frame via common medium
                frame s = from_physical_layer();

                //convert the frame into a packet
                packet p;
                p = s.info;

                //generate a random number
                int r = (rand()+rand())%2;

                //if r = 0, it means that frame is correctly received
                //if r = 1, it means that frame was damaged, therefore a repeat acknowledgment is
sent
                switch(r){
```

```
            case 0: cout<<"\n[RECEIVER]: frame correctly received";

                //pass the packet to network layer
                to_network_layer(p);

                        //send acknowledgement via common medium with next sequence
number
                        ack.kind = ACK;
                        ack.sno = next_seq_no;

                        cout<<"\n[RECEIVER]: sending acknowledgment (sno=" << ack.sno <<
")";
                        common_medium = ack;
                        break;

                case 1:
                cout<<"\n[RECEIVER]: corrupted frame received";
                      //send acknowledgement with previous sequence number via common
medium

                        ack.kind = ACK;
                        ack.sno = (next_seq_no+1)%2;

                        cout<<"\n[RECEIVER]: sending acknowledgment (sno=" << ack.sno
<<")";
                        common_medium = ack;
                        break;
      }
        }
        else
   {
        cout<<"\nreceiver waiting for frame to arrive....."<<show(current_event);
        }
        cout<<"\n-----------------------------------------------------------------";
  }

  char* show(event_type e)
  {
   if(e == ready_to_send)
      return "ready_to_send";
   if(e == frame_arrival)
      return "frame_arrival";
   if(e==frame_send)
      return "frame_send";
   if(e==waiting)
      return "waiting";
```

```
    if(e==time_out)
        return "time_out";
    if(e==frame_resend)
        return "frame_resend";
  }
};


int main()
{
        station S;
        char choice = 'y';
        while(choice =='y' || choice == 'Y')
        {
          S.send();
          Sleep(1000);

          int r = rand()%3;
          if(r==0)
          {
           S.receive();
           cout<<"\nDo you want to continue..? (y/n)";
           cin>>choice;
          }

    }
    cout<<"\nProgram Ends...Press any key to continue.";

    getch();

        return 0;
}
```

F:\Networking\Q3 (Stop n Wait Noisy).exe

```
[SENDER]: Enter data to be sent 4

[SENDER]: sending frame to physical layer  (sno=0)
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 1
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 2
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 3
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 4
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 5
--------------------------------------------------------------------
[SENDER]: Repeat frame sending due to time_out
[SENDER]: sending frame to physical layer  (sno=0)
--------------------------------------------------------------------
[RECEIVER]: receiving frame from physical layer
[RECEIVER]: frame correctly received
[RECEIVER]: sending packet to network layer: 4
[RECEIVER]: sending acknowledgment (sno=1)
--------------------------------------------------------------------
Do you want to continue..? (y/n)y

[SENDER]: Enter data to be sent 3

[SENDER]: sending frame to physical layer  (sno=1)
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 1
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 2
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 3
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 4
--------------------------------------------------------------------
[RECEIVER]: receiving frame from physical layer
[RECEIVER]: frame correctly received
[RECEIVER]: sending packet to network layer: 3
[RECEIVER]: sending acknowledgment (sno=0)
--------------------------------------------------------------------
Do you want to continue..? (y/n)y

[SENDER]: Enter data to be sent 7

[SENDER]: sending frame to physical layer  (sno=0)
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 1
--------------------------------------------------------------------
sender waiting for acknowledgment......timer = 2
--------------------------------------------------------------------
[RECEIVER]: receiving frame from physical layer
[RECEIVER]: corrupted frame received
[RECEIVER]: sending acknowledgment (sno=0)
--------------------------------------------------------------------
Do you want to continue..? (y/n)y
```



Ques 5) Implement Go Back n using sliding window.

:-

```
# include <iostream>
//# include <conio.h>
# include <stdlib.h>
# include <time.h>
# include <math.h>
#include<windows.h>

# define TOT_FRAMES 50
# 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();
 system("pause");
 return 0;

}
```

OUTPUT:

Ques 6) Implement Dijistra's Algorithm to find the shortest path .
:-

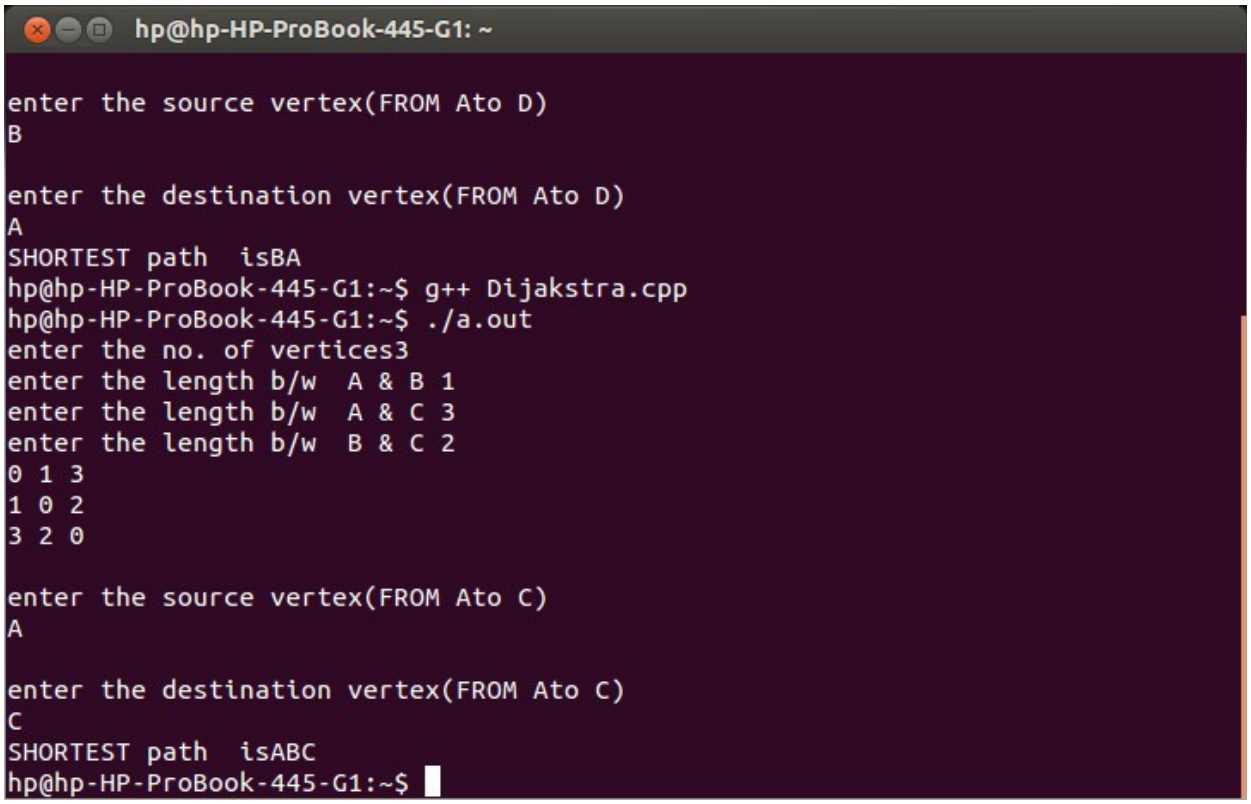
```
#include<iostream>
using namespace std;
#include<stdio.h>
using namespace std;
//#include<conio.h>

using namespace std;
#define INFINITY 10000



void Construct_Graph(int G[][10],int v)
{
	int i;
	for(i=0;i<v;i++)
		for(int j=0;j<v;j++)
		{
			if(i!=j)
			{
				if(G[j][i]==INFINITY)
				{
					cout<<"enter the length b/w  "<<char(i+65)<<" &
"<<char(j+65)<<" ";
```

```
                                        cin>>G[i][j];
                                }
                                else
                                        G[i][j]=G[j][i];
                        }
                }
        for(i=0;i<v;i++)
        {
                for(int j=0;j<v;j++)
                        cout<<G[i][j]<<" ";
                cout<<endl;
        }
}



int minimum(int G[][10],char *type,int *prev,int v)
{
        int pos,min,i,j;
        for( i=0;i<v;i++)
                if(type[i]=='T')
                {
                        min=G[i][prev[i]];
                        break;
                }
        pos=i;
        for( j=i+1;j<v;j++)
        {
                if(type[j]=='T')
                {
                        if(G[j][prev[j]]<min)
                        {
                                min=G[j][prev[j]];
                                pos=j;
                        }

                }
        }
        return pos;
}



void shortest_path(int G[][10],int *prev,char *type,int S,int D,int v)
{
        int k=0;
```

```
        char *path=new char[v];
        type[S]='P';
        path[k]=char(S+65);
        k++;
        while(S!=D)
        {
                for(int i=0;i<v;i++)
                {
                        if((G[S][i]!=-1)&&(type[i]!='P'))
                        {
                                type[i]='T';
                                prev[i]=S;
                        }
                }
                S=minimum(G,type,prev,v);
                type[S]='P';
                path[k]=char(S+65);
                k++;

        }
        cout<<"SHORTEST path  is";
        for(int i=0;i<v;i++)
                cout<<path[i];
        cout<<endl;
        delete path;
}



int main()
{
        int v,S,D,G[10][10],prev[10];
        char V1,type[10];
        cout<<"enter the no. of vertices";
        cin>>v;
        for(int i=0;i<v;i++)
        {
                prev[i]=0;
                type[i]='N';
                for(int j=0;j<v;j++)
                if(i!=j)
                        G[i][j]=INFINITY;
                else
                        G[i][j]=0;
        }
        Construct_Graph(G,v);
```

```
        cout<<"\nenter the source vertex(FROM "<<char(65)<<"to "<<char(v-1+65)<<")\n";
        cin>>V1;
        S=int(V1)-65;
        cout<<"\nenter the destination vertex(FROM "<<char(65)<<"to "<<char(v-
1+65)<<")\n";
        cin>>V1;
        D=int(V1)-65;
        shortest_path(G,prev,type,S,D,v);
   return 0;
}
```

OUTPUT :



F:\Networking\Q7.exe

```
Enter no of vertices :- 6

Enter no of edges :- 8

Enter Cost corresponding to each edge :
Source Node :- 1

Destination Node :- 2

Cost :- 3

Source Node :- 1

Destination Node :- 5

Cost :- 4

Source Node :- 1

Destination Node :- 6

Cost :- 3

Source Node :- 2

Destination Node :- 3

Cost :- 5

Source Node :- 2

Destination Node :- 5

Cost :- 4

Source Node :- 4

Destination Node :- 5

Cost :- 6

Source Node :- 5

Destination Node :- 6

Cost :- 2

Source Node :- 3

Destination Node :- 4

Cost :- 1

Enter initial vertex :- 1
1  2
1  6
1  5
```