COMPUTER NETWORKS PRACTICALS

BY-

Simran Pandey

ROLL NO.-15/1296

B.SC.(Hons) COMPUTER SCIENCE

THIRD SEMESTER

INDEX

1. WAP to implement CRC-12 for noiseless channel.
2. WAP to implement CRC-12 for noisy channel.
3. WAP to implement STOP N WAIT protocol for noiseless channel.
4. WAP to implement STOP N WAIT protocol for noisy

Channel.

5.WAP to implement Selective Repeat Protocol Using SLIDING WINDOW.

6.WAP to implement GO BACK N protocol using SLIDING

WINDOW.

7. WAP to implement DIJISTRA algorithm to compute

shortest path.

Ques 1) Implement CRC-12 for Noiseless channel.

//CRC-12 - Cyclic Redundancy Check.

//Noiseless Channel.

#include<iostream>

using namespace std;

string divide(string , string); //Devision function.

bool error\_detect(string); //Returns 1 if there is an error in message, else returns 0.

int main()

{

string msg, gen, msg1, msg2, remainder1, remainder2;

int genlen, crcbits,msglen;

cout<<"Enter message : ";

cin>>msg;

cout<<"\nEnter Generator : ";

cin>>gen;

msglen=msg.length();

genlen=gen.length();

crcbits=genlen-1;

msg1=msg;

for (int i=0; i<crcbits; i++)

msg1+="0";

cout<<"\nMessage+CRC Bits : "<<msg1;

cout<<"\n msg..."<<msg1;

remainder1=divide(msg1,gen);

cout<<"\n\nREMAINDER 1 : "<<remainder1;

msg2=msg+remainder1;

cout<<"\n\nMessage+Remainder : "<<msg2;

remainder2=divide(msg2,gen);

cout<<"\n\nREMAINDER 2 : "<<remainder2;

cout<<"\n\n";

bool flag=error\_detect(remainder2);

if(flag==1)

cout<<"\n\nERROR DETECTED AT RECEIVER END!!!";

else

cout<<"\n\nNO ERROR DETECTED AT RECEIVER END!!!";

cout<<endl;

system("pause");

return 0;

}

string divide(string newmsg , string gen)

{

string remainder;

int genlen, len;

string check;

genlen=gen.length();

remainder=newmsg;

do

{

string temp;

int i;

if(remainder[0]=='1')

check=gen;

else

{

check="";

for (i=0; i<genlen; i++)

check+="0";

}

for (i=0; i<genlen; i++)

{

if (remainder[i]==check[i])

remainder[i]='0';

else

remainder[i]='1';

}

len=remainder.length();

temp=remainder;

for(i=0; i<len-1; i++)

remainder[i]=temp[i+1];

remainder[i]='\0';

remainder.resize(len-1);

cout<<"\nRemainder : "<<remainder;

}

while(remainder.length()>=genlen);

return remainder;

}

bool error\_detect(string remainder)

{

bool flag=0; //if no error

int len=remainder.length();

for(int i=0; i<len; i++)

{

if(remainder[i]!='0')

{

flag=1;

break;

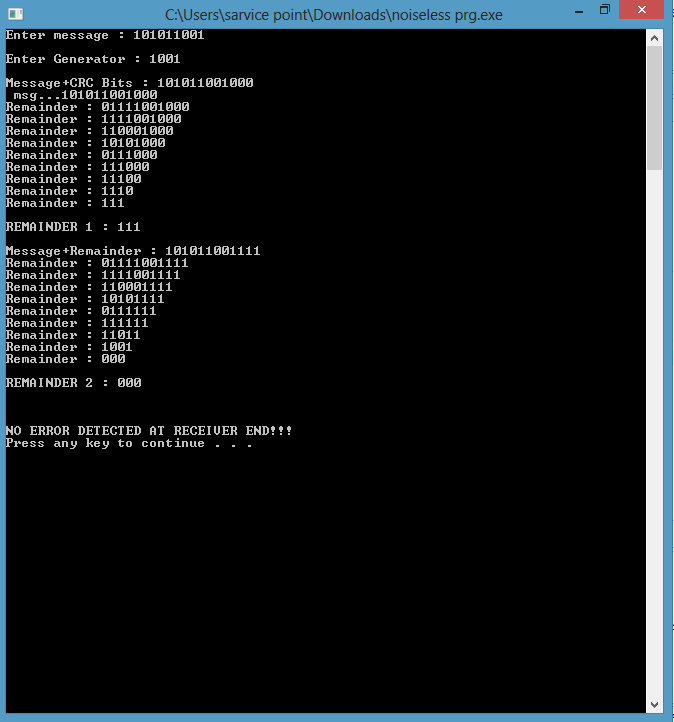
}

}

return flag;

}

OUTPUT-



Ques 2) Implement CRC-12 with Noisy Channel.

#include<iostream>

#include<cstdlib>

using namespace std;

int main()

{

int A[10],B[5],m,n,i,C[10],D[10],E[10],noise,s=0;

cout<<"\n\nEnter length of dividend: ";

cin>>m;

cout<<"\n\nEnter length of divisor: ";

cin>>n;

cout<<"\n\nEnter dividend: "<<endl;

for(i=0;i<m;i++)

{ //takes dividend

cin>>A[i];

C[i]=A[i];

}

cout<<"\n\nEnter divisor: "<<endl;

for(i=0;i<n;i++) //takes divisor

cin>>B[i];

for(i=m;i<m+n-1;i++)

{ //add required no. of 0

A[i]=0;

C[i]=A[i];

E[i]=A[i];

}

m=m+n-1;//increases length of dividend

int j=0,y;

int k=m;

while(k>=n)//length of dividend is greater than length of divisor

{

if(A[j]==B[0])

{

for(i=j,y=0;y<n;i++,y++)//changes dividend

{

if(A[i]==B[y])

A[i]=0;

else

A[i]=1;

}

}

else

{

j++;//if bit of dividend not equal to first bit of divisor

k--;

}

}

// to show the message

for(i=j;i<m;i++)

C[i]=A[i];

/\*cout<<"Message sent is: "<<endl;

for(int i=0;i<m;i++)

cout<<C[i];

cout<<endl;\*/

/\* to pass message through noise or noiseless channel

cout<<" menu "<<endl<<"1. to pass the message through noiseless medium type 0"<<endl

<<"2. to pass the message through noisy medium type 1"<<endl;

cin>>noise;\*/

for(i=0;i<m;i++)

E[i]=C[i];

while(s<10)

{for(i=0;i<m;i++)

C[i]=E[i];

cout<<" \n\nsending message at"<<s<<" time"<<endl;

s++;

noise = rand()%2;

if (noise == 0)

{

cout<<"\n\nmessage is sending through noiseless medium "<<endl;

}

else

{

cout<<"\n\nmessage is sending through noisy medium "<<endl;

int pos = rand()%m;

if (C[pos]==0)

C[pos]=1;

else

C[pos]=0;

//C[i]=A[i];

}

cout<<"\n\nMessage sent is: "<<endl;

for(int i=0;i<m;i++)

cout<<C[i];

cout<<endl;

/\*cout<<endl<<"Enter the message received by receiver: ";

for(i=0;i<m;i++)

{ //takes dividend

cin>>D[i];

}\*/

for(i=0;i<m;i++)

D[i]=C[i];

k=m;j=0;

while(k>=n)//length of dividend is greater than length of divisor

{

if(D[j]==B[0])

{

for(i=j,y=0;y<n;i++,y++)//changes dividend

{

if(D[i]==B[y])

D[i]=0;

else

D[i]=1;

}

}

else

{

j++;//if bit of dividend not equal to first bit of divisor

k--;

}

}

cout<<"\n\ndecoding: ";

int flag=0;

for(i=j;i<m;i++)

{

if(D[i]!=0)

{

flag=1;

break;

}

}

if(flag==1)

cout<<"\n\nERROR in message"<<endl;

else

cout<<"\n\nNO ERROR"<<endl;

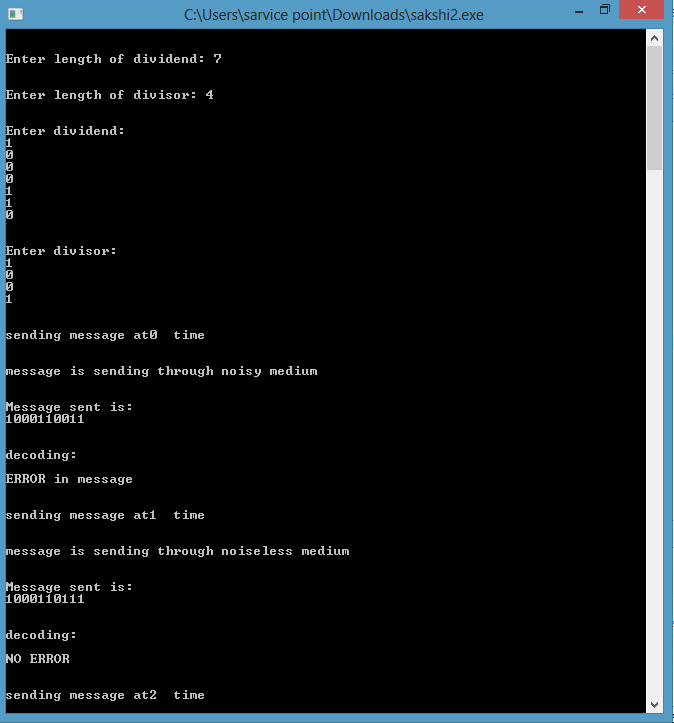
}

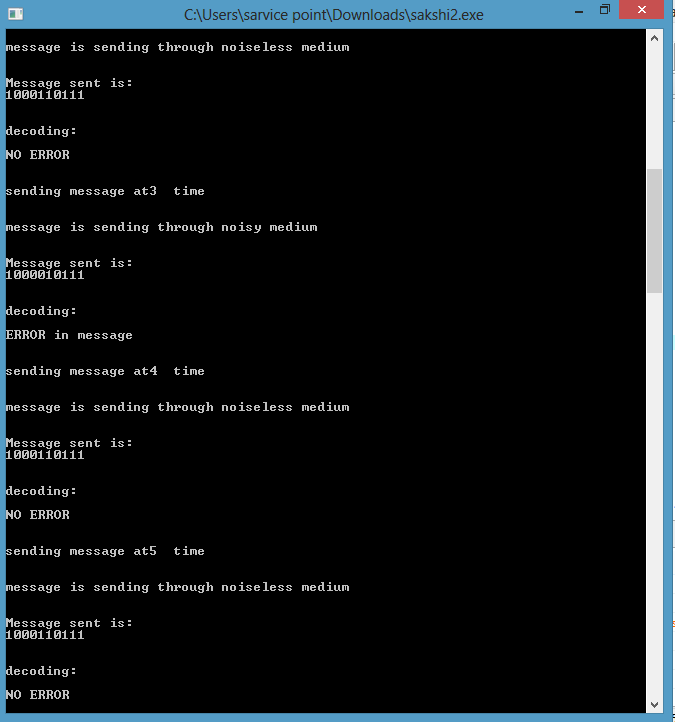
system("pause");

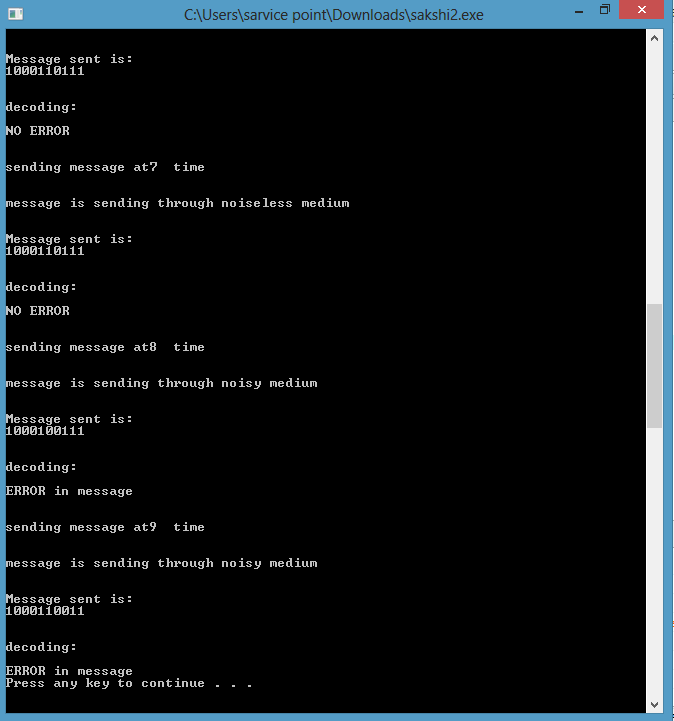
return 0;

}

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;

}

}

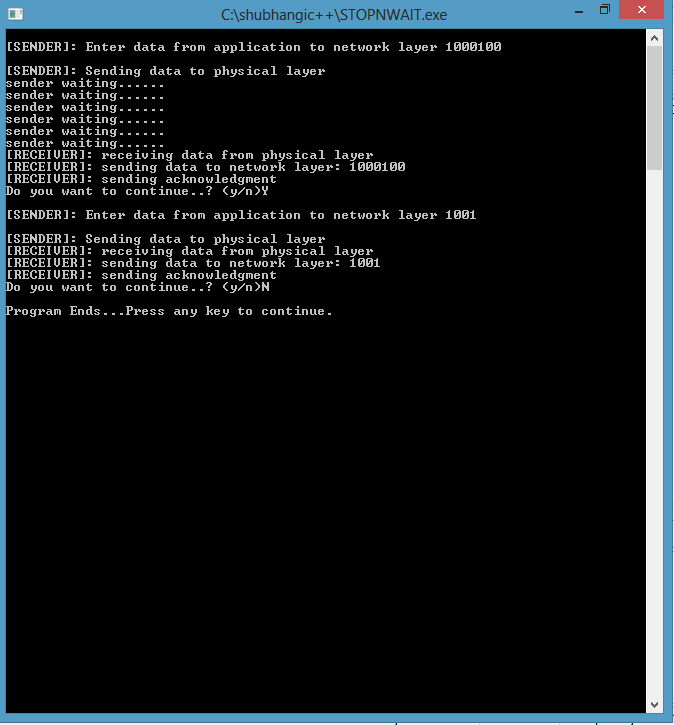
cout<<"\nProgram Ends...Press any key to continue.";

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;

}

}

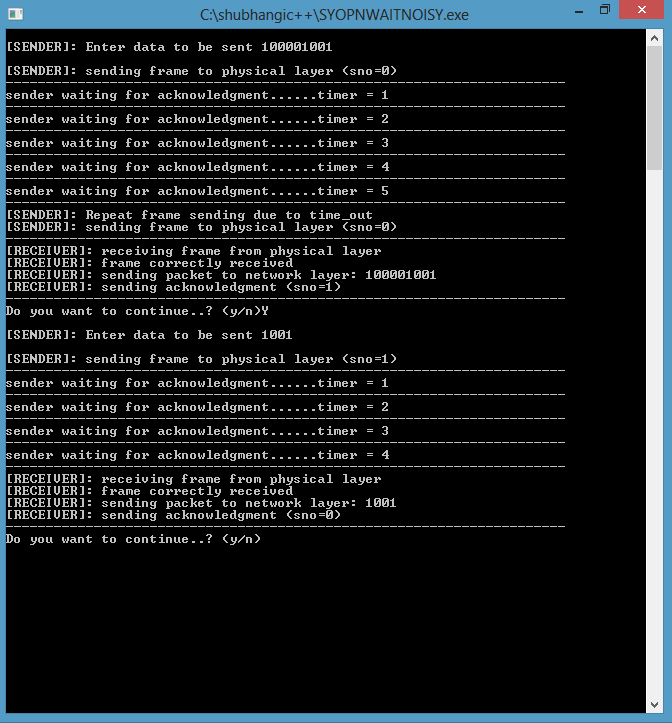
cout<<"\nProgram Ends...Press any key to continue.";

getch();

return 0;

}

OUTPUT-



Ques 5)Simulate Sliding Window Selective Repeat Protocol.

#include<iostream>

using namespace std;

#include<conio.h>

#include<stdlib.h>

#include<time.h>

#include<math.h>

#define TOT\_FRAMES 500

#define FRAMES\_SEND 10

class sel\_repeat

{

private:

int fr\_send\_at\_instance;

int arr[TOT\_FRAMES];

int send[FRAMES\_SEND];

int rcvd[FRAMES\_SEND];

char rcvd\_ack[FRAMES\_SEND];

int sw;

int rw; //tells expected frame

public:

void input();

void sender(int);

void receiver(int);

};

void sel\_repeat::input()

{

int n; //no. of bits for the frame

int m; //no. of frames from n bits

int i;

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(i=0;i<TOT\_FRAMES;i++)

{

arr[i]=t;

t=(t+1)%m;

}

for(i=0;i<fr\_send\_at\_instance;i++)

{

send[i]=arr[i];

rcvd[i]=arr[i];

rcvd\_ack[i]='n';

}

rw=sw=fr\_send\_at\_instance;

sender(m);

}

void sel\_repeat::sender(int m)

{

for(int i=0;i<fr\_send\_at\_instance;i++)

{

if(rcvd\_ack[i]=='n')

cout<<"SENDER : Frame "<<send[i]<<" is sent\n";

}

receiver(m);

}

void sel\_repeat::receiver(int m)

{

time\_t t;

int f;

int j;

int f1;

int a1;

char ch;

srand((unsigned)time(&t));

for(int i=0;i<fr\_send\_at\_instance;i++)

{

if(rcvd\_ack[i]=='n')

{

f=rand()%10;

//if f=5 frame is discarded for some reason

//else frame is correctly recieved

if(f!=5)

{

for(int j=0;j<fr\_send\_at\_instance;j++)

if(rcvd[j]==send[i])

{

cout<<"reciever:Frame "<<rcvd[j]<<" recieved correctly\n";

rcvd[j]=arr[rw];

rw=(rw+1)%m;

break;

}

int j;

if(j==fr\_send\_at\_instance)

cout<<"reciever:Duplicate frame "<<send[i]<<" discarded\n";

a1=rand()%5;

//if al==3 then ack is lost

//else recieved

if(a1==3)

{

cout<<"(acknowledgement "<<send[i]<<" lost)\n";

cout<<"(sender timeouts-->Resend the frame)\n";

rcvd\_ack[i]='n';

}

else

{

cout<<"(acknowledgement "<<send[i]<<" recieved)\n";

rcvd\_ack[i]='p';

}

}

else

{int ld=rand()%2;

//if =0 then frame damaged

//else frame lost

if(ld==0)

{

cout<<"RECEIVER : Frame "<<send[i]<<" is damaged\n";

cout<<"RECEIVER : Negative Acknowledgement "<<send[i]<<" sent\n";

}

else

{

cout<<"RECEIVER : Frame "<<send[i]<<" is lost\n";

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

}

rcvd\_ack[i]='n';

}

}

}

for(int j=0;j<fr\_send\_at\_instance;j++)

{

if(rcvd\_ack[j]=='n')

break;

}

int i=0;

for(int k=j;k<fr\_send\_at\_instance;k++)

{

send[i]=send[k];

if(rcvd\_ack[k]=='n')

rcvd\_ack[i]='n';

else

rcvd\_ack[i]='p';

i++;

}

if(i!=fr\_send\_at\_instance)

{

for(int k=i;k<fr\_send\_at\_instance;k++)

{

send[k]=arr[sw];

sw=(sw+1)%m;

rcvd\_ack[k]='n';

}

}

cout<<"Want to continue :";

cin>>ch;

cout<<"\n";

if(ch=='y')

sender(m);

else

exit(0);

}

int main()

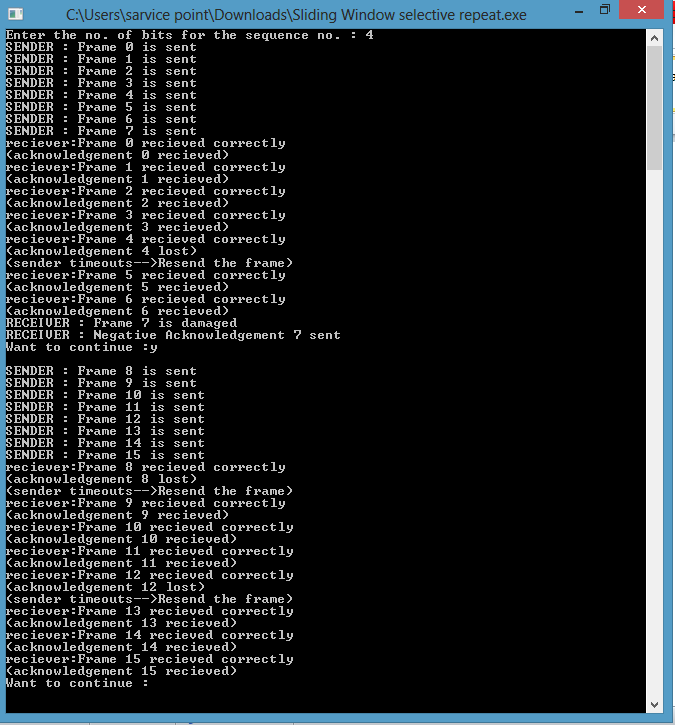
{

sel\_repeat sr;

sr.input();

}

OUTPUT-



Ques 6) 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;

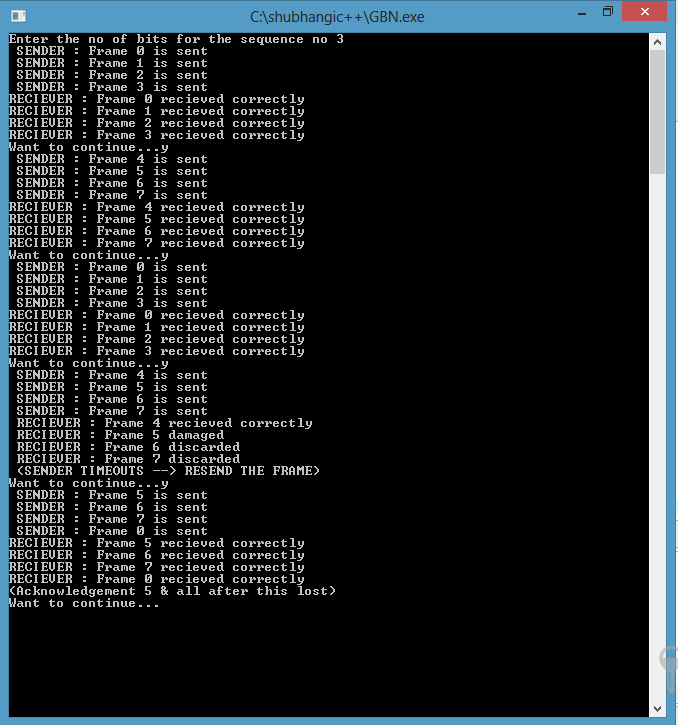
gb.input();

system("pause");

return 0;

}

OUTPUT-



Ques 7).Implement Dijikstra’s Algorithm for Shortest Path Routing.

#include<iostream>

#include<stdio.h>

#include<conio.h>

using namespace std;

#define INFINITY 10000

// CONSTRUCTING THE GRAPH FoR THE NODES

void Con\_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];

}

}

cout<<"\n Constructed matrix is - "<<endl;

for(i=0;i<v;i++)

{

for(int j=0;j<v;j++)

cout<<G[i][j]<<" ";

cout<<endl;

}

}

//FINDING THE MINIMUM DISTANCE BETWEEN THE NODES

int mini(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;

}

// FINDING THE SHORTEST PATH BETWEEN THE SOURCE NODE AND DESTINATION NODE

void Shortest(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= mini(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 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;

}

Con\_Graph(G,v);

cout<<"\n Enter 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(G,prev,type,S,D,v);

return 0;

}

OUTPUT-

