**DELHI TECHNOLOGICAL UNIVERSITY**

Bawana Rd, Shahbad Daulatpur Village, Rohini, New Delhi, Delhi 110042



**COMPUTER NETWORKS (CO 306)**

**LAB FILE**

**SUBMITTED BY: SUBMITTED TO:**

**Ritik Singh Prof. Poonam Rani**

**2K19/CO/319**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME OF EXPERIMENT** | **EXPERIMENT DATE** | **CHECKED ON** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**EXPERIMENT – 1**

**AIM:** Write a program to implement bit stuffing using C++.

**THEORY:**

Data link layer is responsible for something called Framing, which is the division of stream of bits from network layer into manageable units (called frames). Frames could be of fixed size or variable size. In variable-size framing, we need a way to define the end of the frame and the beginning of the next frame.

Bit stuffing is the insertion of non-informative bits into data.

Note: Stuffed bits should not be confused with overhead bits. Overhead bits are non-data bits that are necessary for transmission.

**Applications of Bit Stuffing -**

1. Synchronize several channels before multiplexing
2. Rate-match two single channels to each other
3. Run length limited coding

Bit stuffing technique does not ensure that the sent data is intact at the receiver side (i.e., not corrupted by transmission errors). It is merely a way to ensure that the transmission starts and ends at the correct places.

**Disadvantages of Bit Stuffing** **-**

The code rate is unpredictable, It depends on the data being transmitted.

**Example of bit stuffing** -

Bit sequence: 110101111101011111101011111110 (without bit stuffing)

Bit sequence: 1101011111**0**01011111**0**101011111**0**110 (with bit stuffing)

After 5 consecutive 1-bits, a 0-bit is stuffed. Stuffed bits are marked bold.

**ALGORITHM:**

Here, I and j are two pointers and res is the resultant frame (vector), frame is the original, that is unmodified vector.

1. Start with two pointers (i = j = 0)
2. Create an empty vector
3. while j < frame.size(), do below:
   1. if(frame[j] is not equal to 1)
      1. insert jth bit into res vector
      2. increment i and j;
   2. else
      1. increment j by 1
      2. insert ith bit into res vector
      3. while j < frame.size( ) and jth bit is 1, do below:
         1. if (window\_size is equal to 5)
            1. add extra 0 bit (stuffed bit)
         2. increment j by 1
      4. if (j < frame.size( ))

insert jth bit into res vector

* + 1. i = j + 1;
    2. j = i;

1. return res vector

**CODE:**

#include<iostream>

#include<vector>

using namespace std;

vector<int> bit\_stuffing(vector<int> frame){

if(frame.size() < 5){

return frame;

}

vector<int> ans;

int i,j;

i = 0;

j = i;

while(j<frame.size()){

if(frame[j] != 1){

ans.push\_back(frame[j]);

i++;

j++;

}

else{

j = j + 1;

ans.push\_back(frame[i]);

while(j < frame.size() && frame[j] == 1){

ans.push\_back(frame[j]);

if(j - i + 1 == 5){

ans.push\_back(0);

}

j++;

}

if(j<frame.size())

ans.push\_back(frame[j]);

i = j + 1;

j = i;

}

}

return ans;

}

int main(){

int n;

cout<<"Enter the Number of bits in the frame : ";

cin>>n;

cout<<"Frame Before bit Stuffing : "<<endl;

vector<int> frame(n, 0);

for(int i = 0 ; i < frame.size() ; i++){

cin>>frame[i];

}

vector<int> ans = bit\_stuffing(frame);

cout<<"Frame After Bit Stuffing : "<<endl;

for(int i = 0 ; i < ans.size() ; i++){

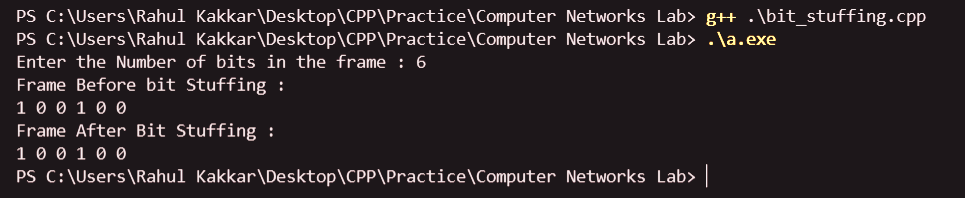
cout<<ans[i]<<" ";

}

cout<<endl;

}

**OUTPUT:**

****

**LEARNING OUTCOME**

• synchronize several channels before multiplexing

• rate-match two single channels to each other

• run length limited coding

**EXPERIMENT – 2**

**AIM:** Write a program to implement byte stuffing using C.

**THEORY:**

In Data Link layer, the stream of bits from physical layer are divided into data frames. The data frames can be of fixed length or variable length. In variable – length framing, the size of each frame to be transmitted may be different. So, a pattern of bits is used as a delimiter to mark the end of one frame and the beginning of the next frame. However, if the pattern occurs in the message, then mechanisms need to be incorporated so that this situation is avoided.

A byte is stuffed in the message to differentiate from the delimiter. This is also called character-oriented framing.

**Byte Stuffing Mechanism**

If the pattern of the flag byte is present in the message byte, there should be a strategy so that the receiver does not consider the pattern as the end of the frame. In character – oriented protocol, the mechanism adopted is byte stuffing.

In byte stuffing, a special byte called the escape character (ESC) is stuffed before every byte in the message with the same pattern as the flag byte. If the ESC sequence is found in the message byte, then another ESC byte is stuffed before it.

**CODE:**

#include<stdio.h>

#include<string.h>

void main()

{

char a[30], fs[50] = " ", t[3], sd, ed, x[3], s[3], d[3], y[3];

int i, j, p = 0, q = 0;

printf("Enter characters to be stuffed:");

scanf("%s", a);

printf("\nEnter a character that represents starting delimiter:");

scanf(" %c", &sd);

printf("\nEnter a character that represents ending delimiter:");

scanf(" %c", &ed);

x[0] = s[0] = s[1] = sd;

x[1] = s[2] = '\0';

y[0] = d[0] = d[1] = ed;

d[2] = y[1] = '\0';

strcat(fs, x);

for(i = 0; i < strlen(a); i++)

{

t[0] = a[i];

t[1] = '\0';

if(t[0] == sd)

strcat(fs, s);

else if(t[0] == ed)

strcat(fs, d);

else

strcat(fs, t);

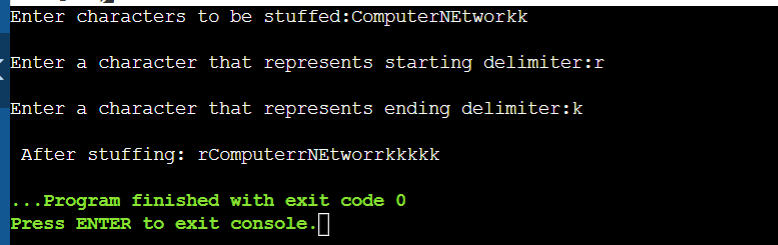
}

strcat(fs, y);

printf("\n After stuffing:%s", fs);

}

**OUTPUT:**



**CONCLUSION:**

Concept of byte stuffing is understood clearly. The program was implemented correctly.

**EXPERIMENT - 3**

**AIM:** Write a program to implement character stuffing.

**THEORY:**

Error correction and error detection are the most prominent features that are to be provided by the application which transfers data from one end to the other. One such mechanism in tracking errors that may add up to the original data during transfer is "Stuffing".

**Stuffing is of three types:**

* Bit stuffing
* Byte stuffing
* Character Stuffing.

**Character Stuffing**

* Each frame starts with the ASCII character sequence DLE STX and ends with the sequence DLE ETX. (Where DLE is Data Link Escape, STX is Start of Text and ETX is End of Text.) This method overcomes the drawbacks of the character count method. If the destination ever loses synchronization, it only must look for DLE STX and DLE ETX characters.
* If binary data is being transmitted, then there exists a possibility of the characters DLE STX and DLE ETX occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used. The sender's data link layer inserts an ASCII DLE character just before the DLE character in the data. The receiver's data link layer removes this DLE before this data is given to the network layer. However, character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary-sized characters.

**Character De-Stuffing**

* De-Stuffing refers to the retrieval of the original data from the received data. This is done by removing the special bit patterns and the stuffed bits.

**ALGORITHM:**

**Character-Stuffing**

*Stuffing (Input)*

1. insert "DLESTX" into the input sequence.

2. Start with the original sequence.

3. for every char C in S: if C = ‘D’ and C+1 = ‘L’ and C+2 = ‘E’: Insert a ‘DLE’ before C. 4. insert "DLE ETX" into the input sequence

5. return the new sequence.

*De-Stuffing (Input)*

1. Neglect the initial bit pattern

2. If a stuffed bit is found, neglect it; otherwise, copy the input to output. 3. Neglect the trailing bit pattern

4. End

**CODE:**

#include <bits/stdc++.h>

using namespace std;

void addStartDelimiter(string &s)

{

s += "DLESTX";

return;

}

void addEndDelimiter(string &s)

{

s += "DLEETX";

return;

}

int main()

{

string in, out;

cout << "Enter the character sequence \n" << ">>> ";

cin >> in;

// Stuffing String

addStartDelimiter(out);

int len = in.length();

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

{

if (i < len && i + 1 < len && i + 2 < len)

if (in[i] == 'd' && in[i + 1] == 'l' && in[i + 2] == 'e') out += "dle";

out += in[i];

}

addEndDelimiter(out);

cout << "After stuffing : " << out << "\n";

// De-stuffing String

in.clear();

out = out.substr(6, out.length() - 12);

for (int i = 0; i < out.length(); i++)

{

if (out[i] == 'd' && out[i + 1] == 'l' && out[i + 2] == 'e') {

i += 5;

in += "dle";

}

else in += out[i];

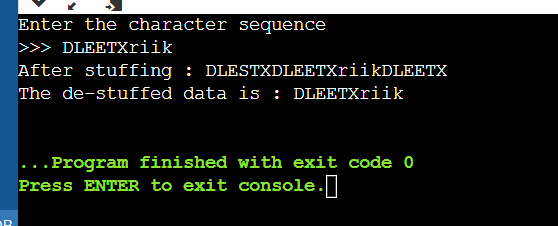
}

cout << "The de-stuffed data is : ";

cout << in << "\n"; return 0;

}

**OUTPUT:**



**LEARNING OUTCOME:**

* Character stuffing is also known as byte stuffing or character-oriented framing and is same as that of bit stuffing but byte stuffing actually operates on bytes whereas bit stuffing operates on bits.
* In byte stuffing, special byte that is basically known as ESC (Escape Character) that has predefined pattern is generally added to data section of the data stream or frame when there is message or character that has same pattern as that of flag byte.
* But receiver removes this ESC and keeps data part that causes some problems or issues. In simple words, we can say that character stuffing is addition of 1 additional byte if there is presence of ESC or flag in text.

**EXPERIMENT - 4**

**AIM:** Write a program to implement cyclic redundancy check (CRC).

**THEORY:**

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.

CRC uses Generator Polynomial which is available on both sender and receiver side. An example generator polynomial is of the form like x3 + x + 1. This generator polynomial represents key 1011. Another example is x2 + 1 that represents key 101.

Redundancy means duplicacy**.** The redundancy bits used by CRC are changed by splitting the data unit by a fixed divisor. The remainder is CRC.

**Qualities of CRC**

* It should have accurately one less bit than the divisor.
* Joining it to the end of the data unit should create the resulting bit sequence precisely divisible by the divisor.

**Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):**

1. The binary data is first augmented by adding k-1 zeros in the end of the data
2. Use modulo-2 binary division to divide binary data by the key and store remainder of division.
3. Append the remainder at the end of the data to form the encoded data and send the same

**Receiver Side (Check if there are errors introduced in transmission)**

Perform modulo-2 division again and if the remainder is 0, then there are no errors.

**CODE:**

#include<iostream>

#include<string>

using namespace std;

string Xor(string a, string b)

{

string result;

for (int i = 1; i < b.length() ; i++)

{

if (a[i] == b[i])

result.append("0");

else

result.append("1");

}

return result;

}

string mod2div( string divident, string divisor)

{

int pick = divisor.length();

string tmp;

string zeros;

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

{

tmp += divident[i];

zeros += "0";

}

while (pick < divident.length())

{

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

tmp = Xor(divisor,tmp) + divident[pick];

else

tmp = Xor(zeros, tmp) + divident[pick];

pick++;

}

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

tmp = Xor(divisor,tmp);

else

tmp = Xor(zeros,tmp);

return tmp;

}

void encode( string data, string key)

{

int key\_len = key.length();

string appended = data;

for ( int i = 0 ; i < key\_len - 1; i++)

{

appended.append("0");

}

string remainder = mod2div(appended, key);

string coded = data + remainder;

cout << "The remainder was : " << remainder << endl;

cout << "The encoded data is : " << coded << endl;

}

bool check\_error(string data, string key)

{

int key\_len = key.length();

string appended = data;

for ( int i = 0 ; i < key\_len - 1; i++)

{

appended.append("0");

}

string remainder = mod2div(appended, key);

for (int i = 0 ; i < remainder.length() ; i++)

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

return false;

return true;

}

int main()

{

string data, key;

cout << "Enter key : ";

cin >> key;

cout << "Enter the data : ";

cin >> data;

encode(data,key);

int flip;

cout << "Enter a bit to flip (to induce error in seq [0, " << data.length() <<"]) : ";

cin >> flip;

if (flip > -1 && flip < data.length())

{

if (data[flip] == '1')

data[flip] = '0';

else

data[flip] = '1';

}

cout << "Checking for transmission errors on received singal : " << data << endl;

if (check\_error(data,key))

cout << "Errors in transmission!" << endl;

else

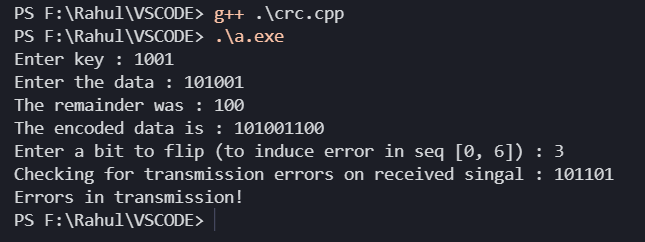
cout << "No Errors found" << endl;

return 0;

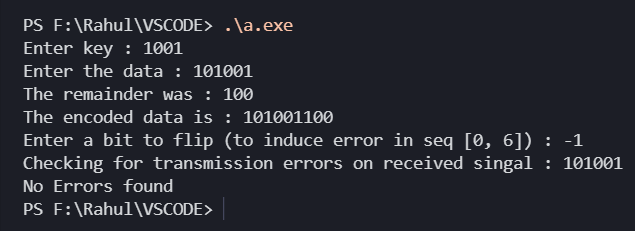
}

**OUTPUT:**

**With error at receiver’s end:**

****

**Without error at receiver’s end:**

****

**LEARNING OUTCOME:**

* A Cyclic redundancy check (CRC) is a rather unusual but clever method that does error checking via polynomial division. It is based on the fact that, under certain conditions, division of an arbitrary polynomial by another, fixed polynomial will almost always yield a nonzero remainder.
* CRCs are popular because they are simple to implement in binary hardware, easy to analyse mathematically, and particularly good at detecting common errors caused by noise in transmission channels.

**EXPERIMENT - 5**

**AIM:** Write a program to implement Stop and wait protocol using C++.

**THEORY:**

It is the simplest flow control method. In this, the sender will transmit one frame at a time to the receiver. The sender will **stop and wait** for the acknowledgement from the receiver.

This time (i.e., the time joining message transmitting and acknowledgement receiving) is the sender’s waiting time, and the sender is idle during this time.

When the sender gets the acknowledgement (ACK), it will send the next data packet to the receiver and wait for the disclosure again, and this process will continue as long as the sender has the data to send.

While sending the data from the sender to the receiver, the data flow needs to be controlled. If the sender is transmitting the data at a rate higher than the receiver can receive and process it, the data will get lost.

The Flow-control methods will help in ensuring that the data doesn't get lost. The flow control method will check that the senders send the data only at a rate that the receiver can receive and process.

**Advantages:**

The main advantage of stop & wait protocols is their accuracy. The next frame is transmitted only when the first frame is acknowledged. So, there is no chance of the frame being lost.

**Disadvantages:**

The drawback of this approach is that it is inefficient. It makes the transmission process slow. An individual frame travels from source to destination in this method, and a single acknowledgement travels from destination to source.

## **Features**

The features of Stop and Wait Protocol are as follows −

* It is used in Connection-oriented communication.
* It offers error and flows control.
* It can be used in data Link and transport Layers.
* Stop and Wait ARQ executes Sliding Window Protocol with Window Size 1.

**CODE:**

#include<iostream>

#include <time.h>

#include <cstdlib>

#include<ctime>

#include <unistd.h>

using namespace std;

class timer {

   private:

   unsigned long begTime;

   public:

   void start() {

   begTime = clock();

   }

unsigned long elapsedTime() {

   return ((unsigned long) clock() - begTime) / CLOCKS\_PER\_SEC;

  }

  bool isTimeout(unsigned long seconds) {

   return seconds >= elapsedTime();

   }

};

int main()

{

int frames[] = {1,2,3,4,5,6,7,8,9,10};

unsigned long seconds = 5;

srand(time(NULL));

timer t;

cout<<"Sender has to send frames : ";

for(int i=0;i<10;i++)

    cout<<frames[i]<<" ";

cout<<endl;

int count = 0;

bool delay = false;

cout<<endl<<"Sender\t\t\t\t\tReceiver"<<endl;

do

{

    bool timeout = false;

    cout<<"Sending Frame : "<<frames[count];

    cout.flush();

    cout<<"\t\t";

    t.start();

    if(rand()%2)

    {

        int to = 24600 + rand()%(64000 - 24600)  + 1;

        for(int i=0;i<64000;i++)

            for(int j=0;j<to;j++) {}

    }

    if(t.elapsedTime() <= seconds)

    {

        cout<<"Received Frame : "<<frames[count]<<" ";

        if(delay)

        {

            cout<<"Duplicate";

            delay = false;

        }

        cout<<endl;

        count++;

    }

    else

    {

        cout<<"---"<<endl;

        cout<<"Timeout"<<endl;

        timeout = true;

    }

    t.start();

    if(rand()%2 || !timeout)

    {

        int to = 24600 + rand()%(64000 - 24600)  + 1;

        for(int i=0;i<64000;i++)

            for(int j=0;j<to;j++) {}

        if(t.elapsedTime() > seconds )

        {

            cout<<"Delayed Ack"<<endl;

            count--;

            delay = true;

        }

        else if(!timeout)

            cout<<"Acknowledgement : "<<frames[count-1]<<endl;

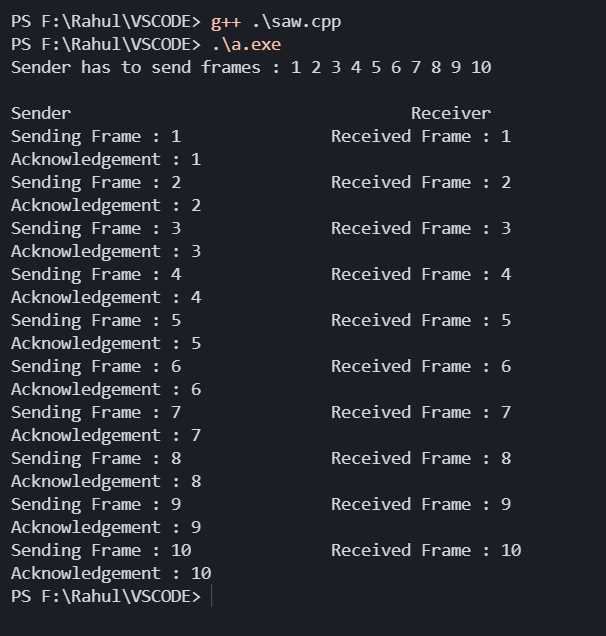
    }

}while(count!=10);

return 0;

}

**OUTPUT:**

****

**LEARNING OUTCOME:**

* Concept of Stop and Wait Protocol is understood clearly, with all the disadvantages and applications. The program was implemented correctly, passed all test and edge cases.
* Advantage of stop & wait protocols is their accuracy. The next frame is transmitted only when the first frame is acknowledged. So, there is no chance of the frame being lost.
* It makes the transmission process slow. An individual frame travels from source to destination in this method, and a single acknowledgement travels from destination to source.

**EXPERIMENT - 6**

**AIM:** Write a program to implement Go Back N protocol using C++.

**THEORY:**

In Go-Back-N ARQ, **N** is the sender's window size. Suppose we say that Go-Back-3, which means that the three frames can be sent at a time before expecting the acknowledgment from the receiver.

It uses the principle of protocol pipelining in which the multiple frames can be sent before receiving the acknowledgment of the first frame. If we have five frames and the concept is Go-Back-3, which means that the three frames can be sent, i.e., frame no 1, frame no 2, frame no 3 can be sent before expecting the acknowledgment of frame no 1.

In Go-Back-N ARQ, the frames are numbered sequentially as Go-Back-N ARQ sends the multiple frames at a time that requires the numbering approach to distinguish the frame from another frame, and these numbers are known as the sequential numbers.

The number of frames that can be sent at a time totally depends on the size of the sender's window. So, we can say that 'N' is the number of frames that can be sent at a time before receiving the acknowledgment from the receiver.

If the acknowledgment of a frame is not received within an agreed-upon time period, then all the frames available in the current window will be retransmitted. Suppose we have sent the frame no 5, but we didn't receive the acknowledgment of frame no 5, and the current window is holding three frames, then these three frames will be retransmitted.

**CODE:**

#include<iostream>

#include<ctime>

#include<cstdlib>

using namespace std;

int main()

{

int nf,N;

int no\_tr=0;

srand(time(NULL));

cout<<"Enter the number of frames : ";

cin>>nf;

cout<<"Enter the Window Size : ";

cin>>N;

int i=1;

while(i<=nf)

{

int x=0;

for(int j=i;j<i+N && j<=nf;j++)

{

cout<<"Sent Frame "<<j<<endl;

no\_tr++;

}

for(int j=i;j<i+N && j<=nf;j++)

{

int flag = rand()%2;

if(!flag)

{

cout<<"Acknowledgment for Frame "<<j<<endl;

x++;

}

else

{ cout<<"Frame "<<j<<" Not Received"<<endl;

cout<<"Retransmitting Window"<<endl;

break;

}

}

cout<<endl;

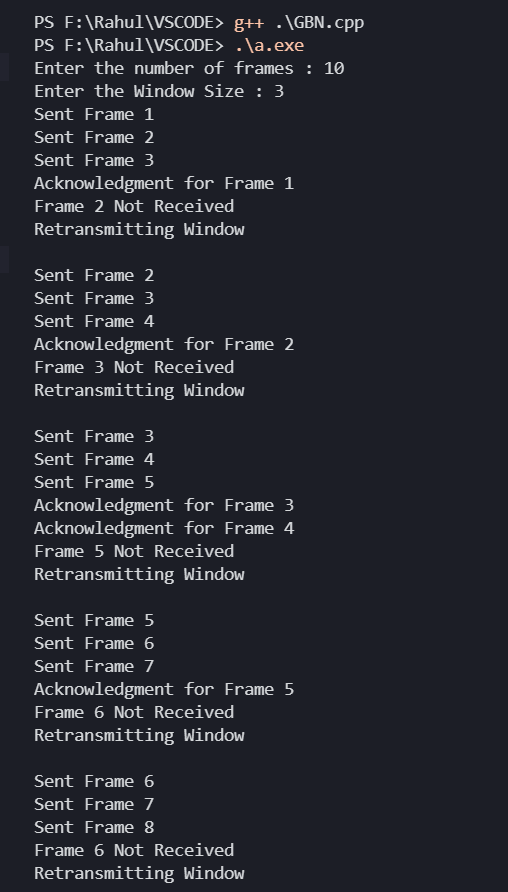
i+=x;

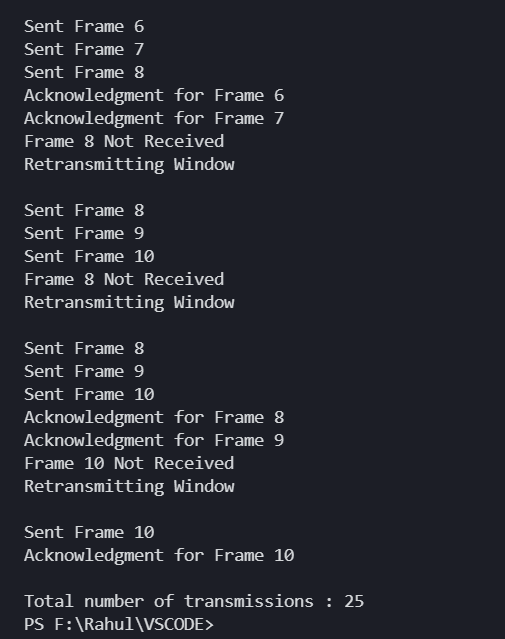
}

cout<<"Total number of transmissions : "<<no\_tr<<endl;

return 0;

}

**OUTPUT:**

****

**LEARNING OUTCOME:**

Concept of Go-Back-N is understood clearly, with all the disadvantages and advantages. The program was implemented correctly, passed all test and edge cases.