SRI VASAVI ENGINEERING COLLEGE (Autonomous)

## PEDATADEPALLI, TADEPALLIGUDEM.





*This is to certify that this is a bonafide record of Practical Work done in Computer Networks LAB by Mr./Miss \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_bearing RollNo \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_of* ***CSE*** *Branch of* ***VI Semester*** *during the academic year* ***2022 -23****.*

No. of Experiments Done: 10

Faculty In charge of the Laboratory Head of the Department

EXTERNAL EXAMINER

**INDEX**

|  |  |  |
| --- | --- | --- |
| S.NO | Experiment | Page. No |
| 1 | Study of basic network commands and Network configuration commands.  1.Ping 2.Tracert/Traceroute  3.Ipconfig/ifconfig 4.Hostname 5.Nslookup 6.Netstat | 3-5 |
| 2 | Construct Detecting error using CRC-CCITT. | 6-10 |
| 3 | Implementing of Bit Stuffing. | 11-12 |
| 4 | Implementation of Character Stuffing. | 13-14 |
| 5 | Implementation of stop and wait protocol. | 15-16 |
| 6 | Implementation of Dijkstra’s algorithm. | 17-19 |
| 7 | Implementation Distance vector algorithm. | 20-25 |
| 8 | Implementation of Congestion control using leaking bucket algorithms. | 26-27 |
| 9 | Implementation using Socket TCP both client and server programs. | 28-30 |
| 10 | Implementation using Socket UDP both client and server programs. | 31-34 |

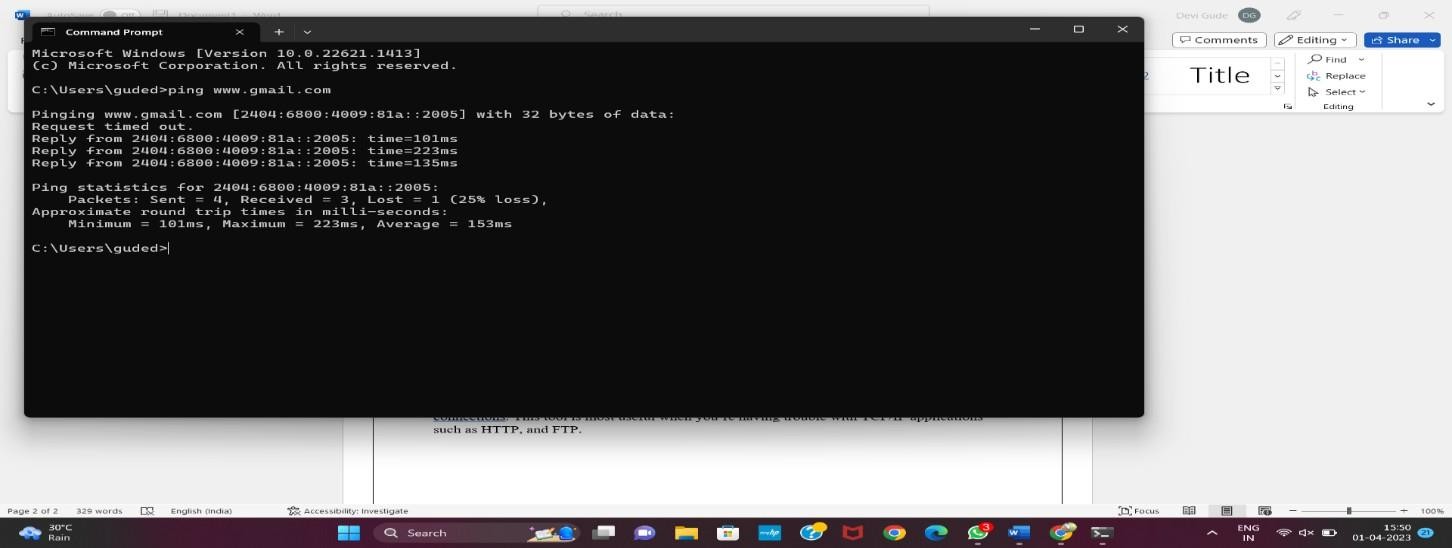
**Task 1 : Study of basic network commands and Network configuration commands.**

**1.Pin 2.Tracert/Traceroute 3.Ipconfig/ifconfig 4.Hostname**

**5.Nslookup 6.Netstat**

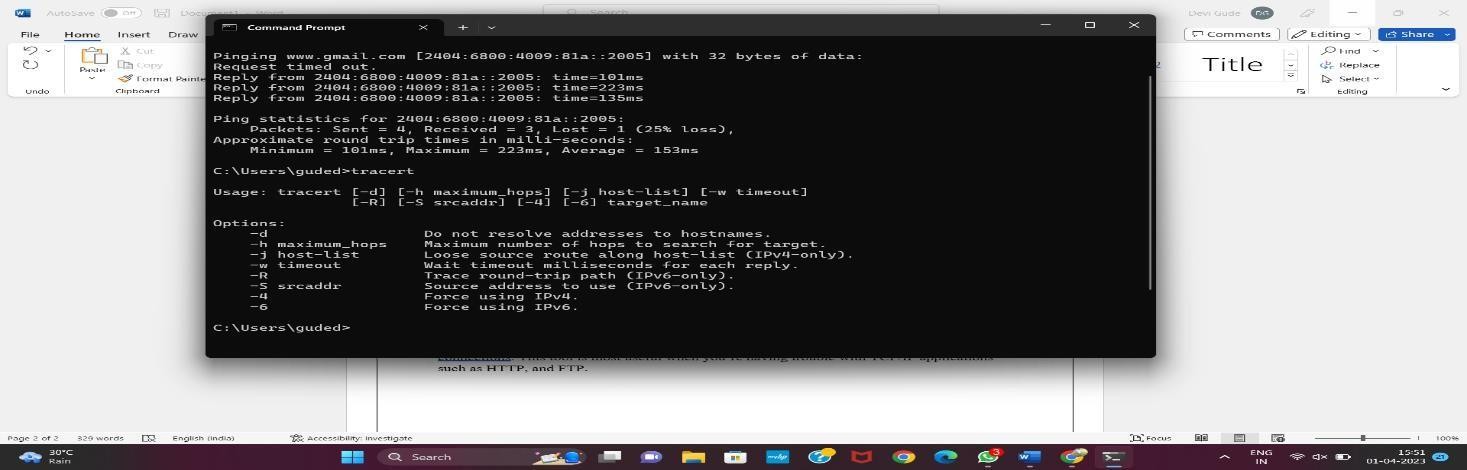
1. **Ping:** ping is the most basic TCP/IP command ,and it’s the same as placing a phone call to your best friend. You pick up your telephone and dial a number, expecting your best friend to reply with “Hello” on the other end. Computers make phone calls to each other over a network by using a ping command. The Ping commands main purpose is to place a phone call to another computer on the network, and request an answer. Ping has 2 options it can us to place a phone call to another computer on the network. It can use the computers name or IP address.

Output:



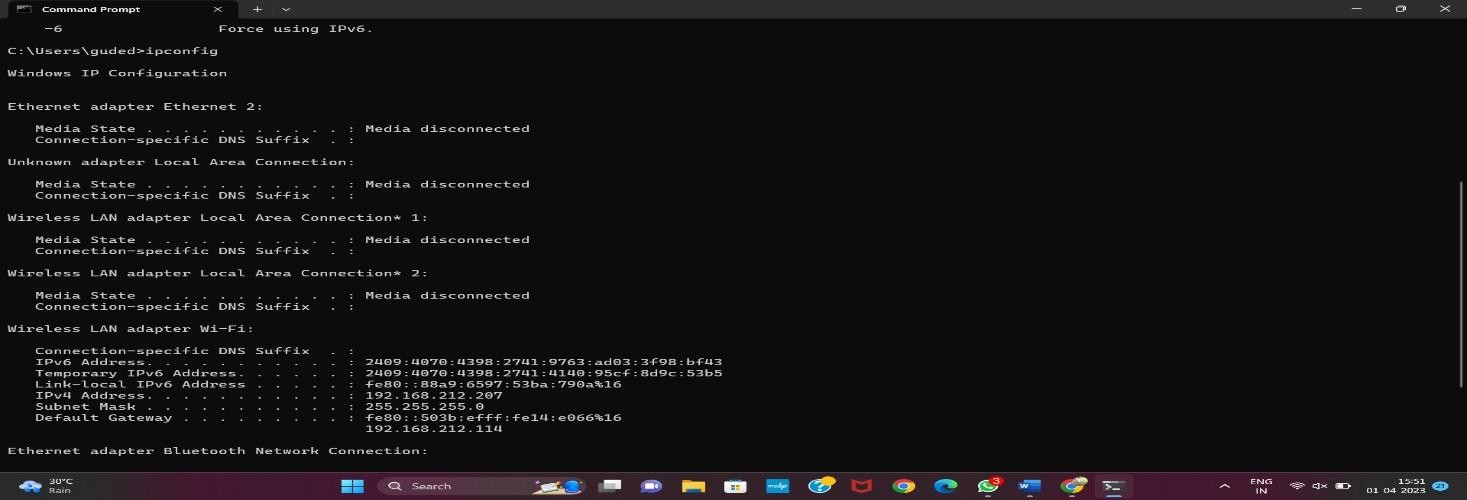
1. **Tracert/Traceroute:** The tracert command displays a list of all the routers that a packet has to go through to get from the computer where tracert is run to any other computer on the internet.

Output:

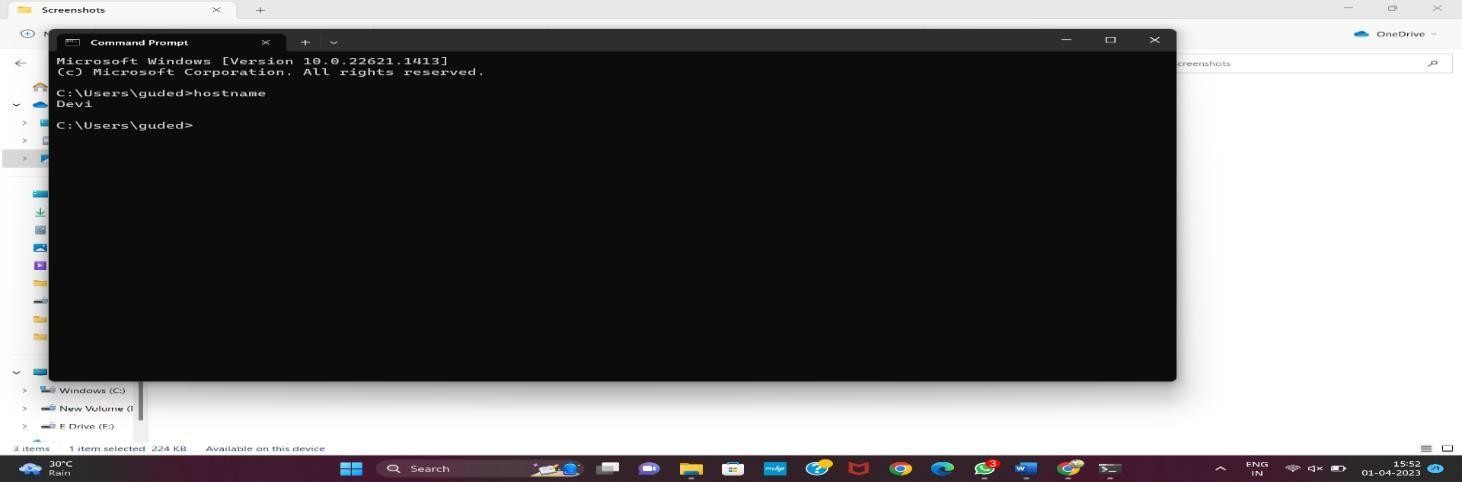


1. **Ipconfig/ifconfig:** The ipconfig command displays information about the host computer TCP/IP configuration.

Output:

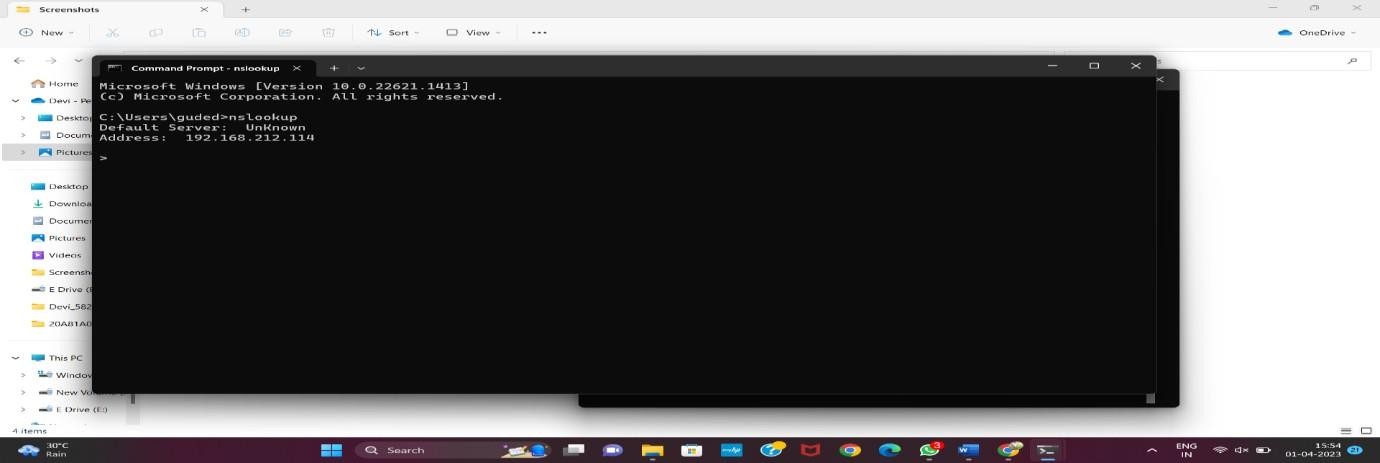


1. **Hostname:** This is the simplest of all TCP/IP commands. It simply displays the name of your computer. Output:



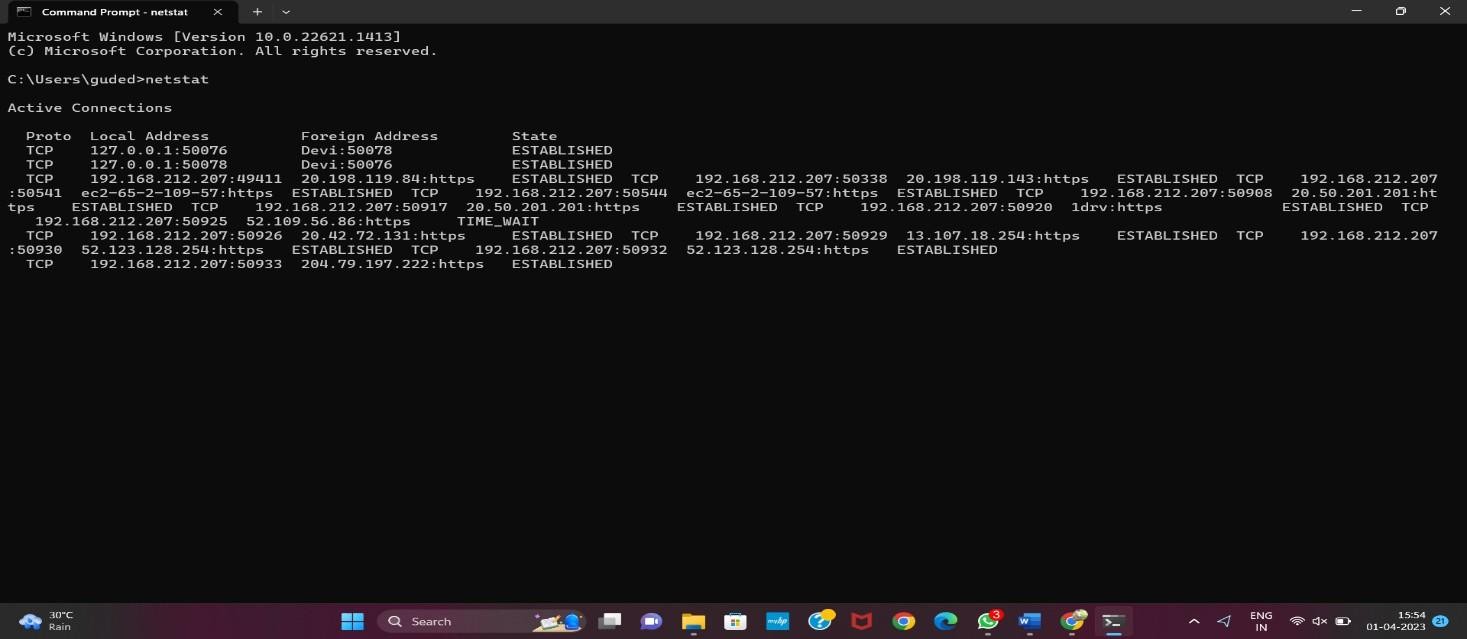
1. **Nslookup**: Nslookup is used for diagnosing DNS problems. If you can access a resource by specifying an IP address but not it’s DNs you have a DNS problem.

Output:



1. **Netstat:** Netstat displays a variety of statistics about a computers active TCP/IP connections. This tool is most useful when you’re having trouble with TCP/IP applications such as HTTP, and FTP.

Output:



# Task2:Construct Detecting error using CRC-CCITTDescription.

The Data Link Layer is the second layer in the OSI model, above the Physical Layer, which ensures that the error free data is transferred between the adjacent nodes in the network. It breaks the datagram passed down by above layers and converts them into frames ready for transfer. This is called Framing. It provides two main functionalities

Reliable data transfer service between two peer network layers

Flow Control mechanism, which regulates the flow of frames such that datacongestion is not there at slow receivers due to fast senders.

There are two basic strategies for dealing with errors. One way is to include enough redundant information (extra bits are introduced into the data stream at thetransmitter on a regular and logical basis) along with each block of data sent to enable thereceiver to deduce what the transmitted character must have been. The other way is to include only enough redundancy to allow the receiver to deduce that error has occurred, but not which error has occurred and the receiver asks for a retransmission. The former strategy uses Error- Correcting Codes and latter uses Error-detecting Codes.

CRC method can detect a single burst of length n, since only one bit per column will be changed, a burst of length n+1 will pass undetected, if the first bit is inverted, thelast bit is inverted and all other bits are correct. If the block is badly garbled by a long burst or by multiple shorter burst, the probability that any of the n columns will have thecorrect parity that is 0.5. so the probability of a bad block being expected when it shouldnot be 2 power(-n). This scheme some times known as Cyclic Redundancy Code Algorithm for computing checksum:

Let ‘r’ be the degree of G(x). Append ‘r’ to the lower end of the frame so that itcontains (m + r) bits. Divide M(x) by G(x) using MOD-2 division.

Subtract the remainder from M(x) using MOD-2 subtraction. The result is the check summed frame to be transmitted.

Eg: frame = 1101011011

G(x) = x4 +x +1 = 10011

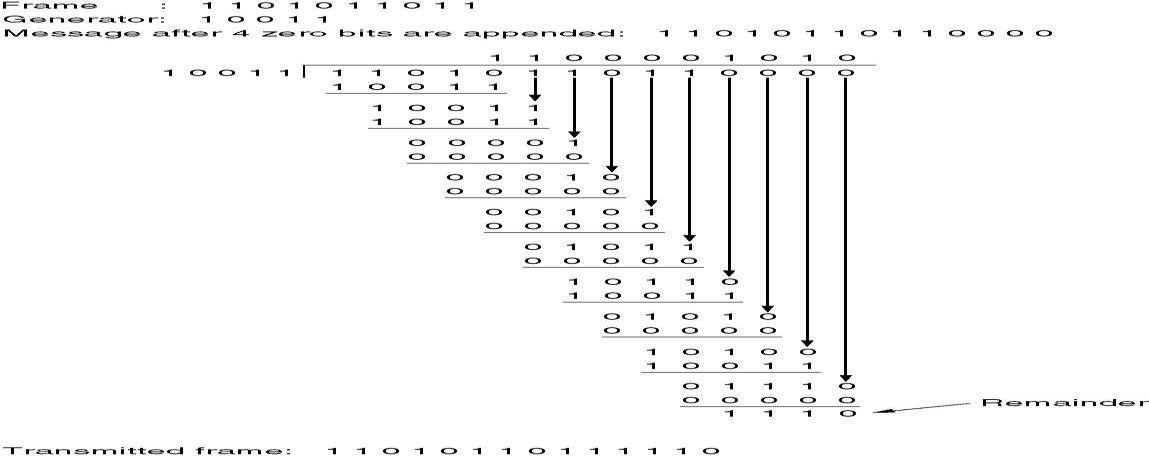
è degree = 4

Therefore, frame = 1101011011 + 0000

èM(x) = 11010110110000

Commonly used divisor polynomials are:

CRC 12: x12 + x11 + x3 + x2 + x + 1CRC 16 : x16 + x15 + x2 + 1 CRC CCITT : x16 + x12 + x5



**Program:** #include<stdio.h>

#include <string.h>

#define N strlen(g)

char t[28],cs[28],g[28];

int a,e,c,b;

void xo(){

for(c=1;c<N;c++) cs[c]=((cs[c]==g[c])?'0':'1');

}

void crc()

{

for(e=0;e<N;e++) cs[e]=t[e];

do{

if(cs[0]=='1') xo();

for(c=0;c<N-1;c++) cs[c]=cs[c+1];

cs[c]=t[e++];

}while(e<=a+N);

}

int main()

{

int flag=0;

do{

printf("\n 1.crc12\n 2.crc16\n 3.crc ccit \n 4.exit \n \n Enter your option.");

scanf("%d",&b);

switch(b)

{

case 1:strcpy(g,"1100000001111");

break;

case 2:strcpy(g,"11000000000000101");

break;

case 3:strcpy(g,"10001000000100001");

break;

case 4:return 0;

}

printf("\n enter data:");

scanf("%s",t);

printf("\n \n");

printf("\n generating polynomial:%s",g);

a=strlen(t);

for(e=a;e<a+N-1;e++) t[e]='0';

printf("\n \n");

printf("modified data is:%s",t); printf("\n \n"); crc(); printf("checksum is:%s",cs); for(e=a;e<a+N1;e++)

t[e]=cs[e-a];

printf("\n \n");

printf("\n final codeword is : %s",t); printf("\n \n");

printf("\ntest error detection 0(yes) 1(no)?:");

scanf("%d",&e); if(e==0)

{

do{

printf("\n\tenter the position where error is to be inserted:"); scanf("%d",&e);

}

while(e==0||e>a+N-1); t[e-1]=(t[e-1]=='0')?'1':'0';

printf("\n \n"); printf("\n\terroneous data:%s\n",t);

}

crc();

for(e=0;(e<N-1)&&(cs[e]!='1');e++);

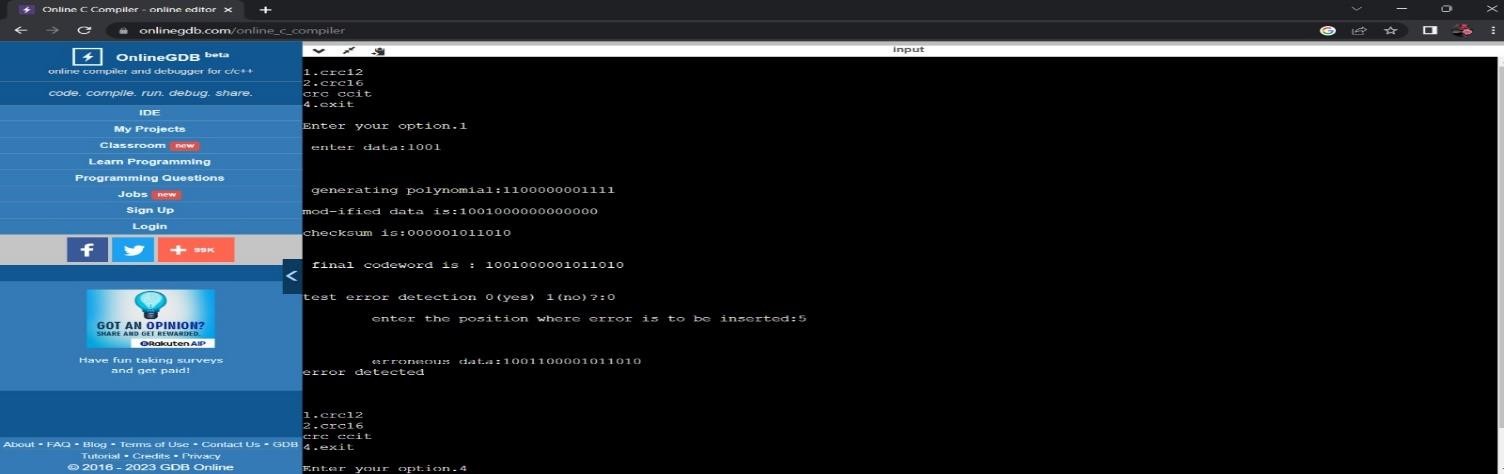
if(e<N-1)

printf("error detected\n\n"); else printf("\n no error detected \n\n"); printf("\n ");

}while(flag!=1);

}

Output:



**Task 3: Program for implementing the data link layer framing method for Bit Stuffing.**

**Description:**

The Bit stuffing technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character. In this each frame begins and ends with a special bit pattern, 01111110 called a flag byte. When ever the sender’s data link encounters five consecutive ones in the data. It automatically stuffs a ‘0’ bit into the out going bit stream. This is called “bit stuffing”.

3) Bit Sequence

#include<stdio.h>

int main()

{

int i=0,len\_ip=0,count;

int len\_op=0;

int j=0,s=0,xxx=0,five=0;

char ip[' '],op[' '],decode\_op[' '];

char pre\_post[]={'0','1','1','1','1','1','1','0'};

printf("Enter input bitsequence: \n");

scanf("%s",ip);

for(i=0;ip[i]!=0;i++) len\_ip++;

/\*Stuffing\*/

for(i=0;i<8;i++) op[i]=pre\_post[i];

count=i;

do

{

if(ip[j]=='1') five++;

else five=0;

op[count]=ip[j];

count++;

j++;

if(five==5)

{

op[count++]='0';

five=0;

}

}while(j<len\_ip);

for(s=count,xxx=0;s<(count+8);s++,xxx++)

{

op[s]=pre\_post[xxx];

}

printf("Output\n");

printf(" ------ ");

printf("\nStuffedBitSequence is:");

op[s]=0;

puts(op);

/\*Destuffing\*/

j=0;

five=0;

len\_op=count+8;

i=8;

do

{

decode\_op[j++]=op[i];

if(op[i]=='1') five++;

else five=0;

if(five==5)

{

i++;

five=0;

}

len\_op--;

i++;

}while((len\_op-8)!=0);

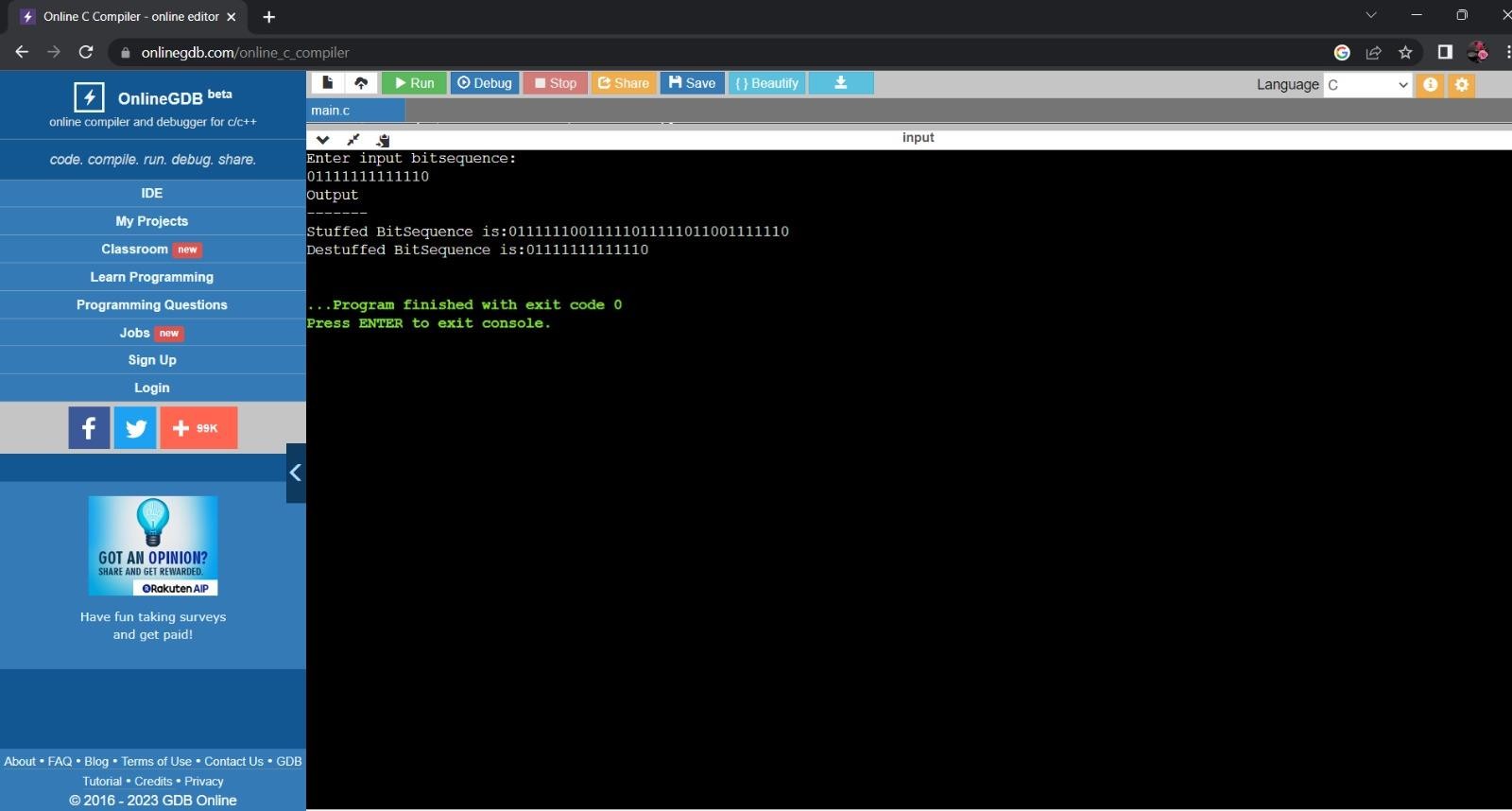
decode\_op[len\_ip]=0;

printf("Destuffed BitSequence is:");

puts(decode\_op);

}

**Output:**



# 

# Task 4 : Program for implementing the data link layer framing method for Character Stuffing.

**Description:**

In character stuffing method, the frame is delimited by DLE STX and DLE ETX. It may easily happen that the characters for DLE STX or DLE ETX occur in the data, which will interface with the framing. One way to solve this problem is to have the sender’s data link layer insert an ASCII DLE character just before each “accidental” DLE character in the data. The data link layer on the receiving end removes the DLE before the data is given to the network layer. This technique is called “Character Stuffing”.

**Program:**

#include<stdio.h>

#include<string.h>

int main()

{

char source[' '],char\_stuff[' '],char\_destuff[' '];

int i=0,j=0,k=0;

printf("Enter plain text");

gets(source);

char\_stuff[0]='d', char\_stuff[1]='l' , char\_stuff[2]='e', char\_stuff[3]='s' , char\_stuff[4]='t', char\_stuff[5]='x';

j=6;

for(k=0;k<strlen(source);)

{

if(source[k]=='d'&&source[k+1]=='l'&&source[k+2]=='e')

{

char\_stuff[j++]='d';char\_stuff[j++]='l';char\_stuff[j++]='e';

char\_stuff[j++]='d';char\_stuff[j++]='l';char\_stuff[j++]='e';

k+=3;

}

else{

char\_stuff[j++]=source[k++];

}

}

char\_stuff[j++]='d';char\_stuff[j++]='l';char\_stuff[j++]='e';

char\_stuff[j++]='s';char\_stuff[j++]='t';char\_stuff[j++]='x';

char\_stuff[j]=0;

printf("After character stuffing:");

puts(char\_stuff);

j=0;

for(i=6;i<(strlen(char\_stuff)-6);)

{

if(char\_stuff[i]=='d'&&char\_stuff[i+1]=='l'&&char\_stuff[i+2]=='e')

{

char\_destuff[j++]='d'; char\_destuff[j++]='l'; char\_destuff[j++]='e';

i=i+6;

}

else{

char\_destuff[j++]=char\_stuff[i++];

}

}

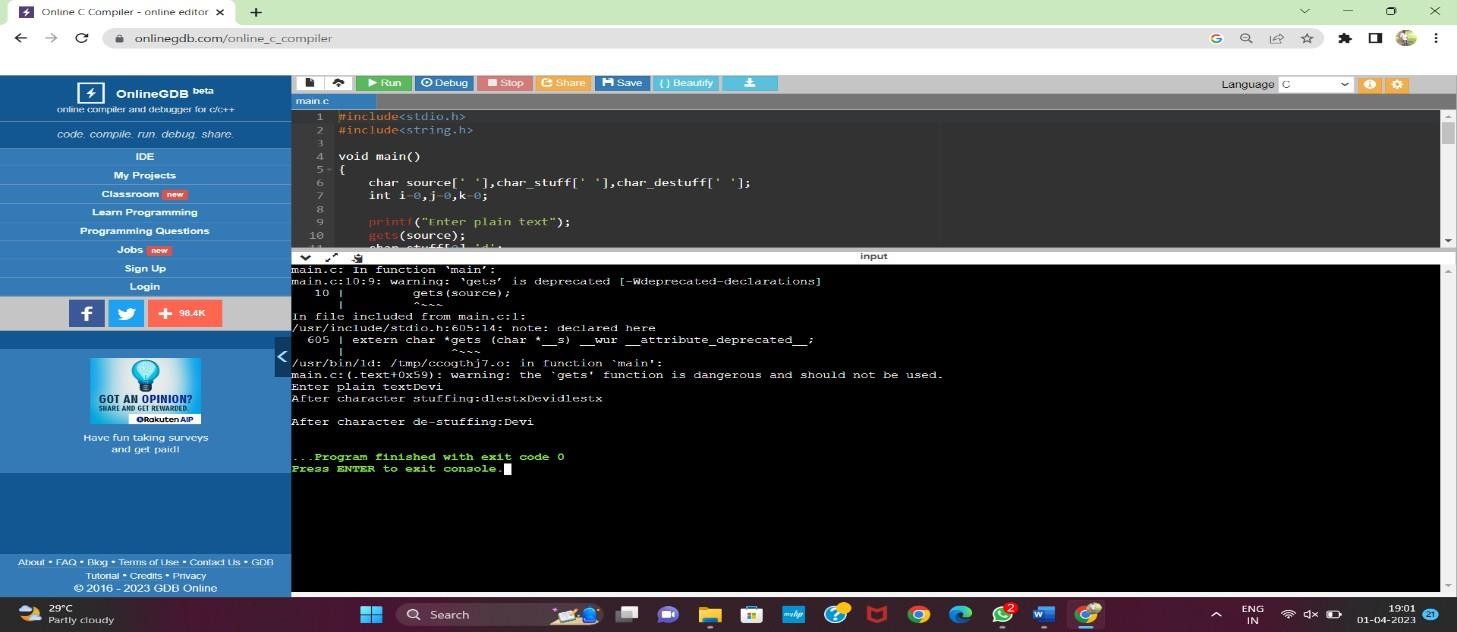
char\_destuff[j]=0;

printf("\nAfter character de-stuffing:");

puts(char\_destuff);

}

**Output:**



# Task 5 : Implementation of stop and wait protocol. Description:

The sender must never transmit a new frame until undone has been fetched from physical layer, then the new one overwrites the old one. The general solution to this problem is to have the receiver send a little dummy frame to the sender which, In effect gives the sender permission to transmit the next frame. Protocol in which in-effect dummy frame back to the sender which in effect gives the sender permission to transmitthe next frame. Protocols in which sender sends one frame and then waits for acknowledgement before proceeding are called stop and wait protocols.

1. Start with the window size of 1 from the transmitting (Source) node
2. After transmission of a frame the transmitting (Source) node waits for a reply(Acknowledgement) from the receiving (Destination) node.
3. If the transmitted frame reaches the receiver (Destination) without error, the receiver(Destination) transmits a Positive Acknowledgement.
4. If the transmitted frame reaches the receiver (Destination) with error, the receiver(Destination) do not transmit acknowledgement.
5. If the transmitter receives a positive acknowledgement it transmits the next frame ifany. Else if the transmission timer expires, it retransmits the same frame again.
6. If the transmitted acknowledgment reaches the Transmitter (Destination) withouterror, the Transmitter (Destination) transmits the next frame if any.
7. If the transmitted frame reaches the Transmitter (Destination) with error, theTransmitter (Destination) transmits the same frame.
8. This concept of the Transmitting (Source) node waiting after transmission for a replyfrom the receiver is known as STOP and WAIT.

**Program:**

#include<stdio.h>

#include<string.h>

# #include<windows.h>

# void main(){

# int n,i,seq,t1=0,t2=0;

# struct tm \*ptr;

# time\_t lt;

# char f[' '];

# printf("enter no of frames : \n");

# scanf("%d",&n);

# for(i=0;i<n;i++){

# printf("enter sequence\n");

# scanf("%d",&seq);

# printf("enter frame");

# scanf("%s",&f);

# lt = time(NULL);

# ptr = localtime(&lt);

# printf(asctime(ptr));

# sleep(5);

# lt = time(NULL);

# ptr = localtime(&lt);

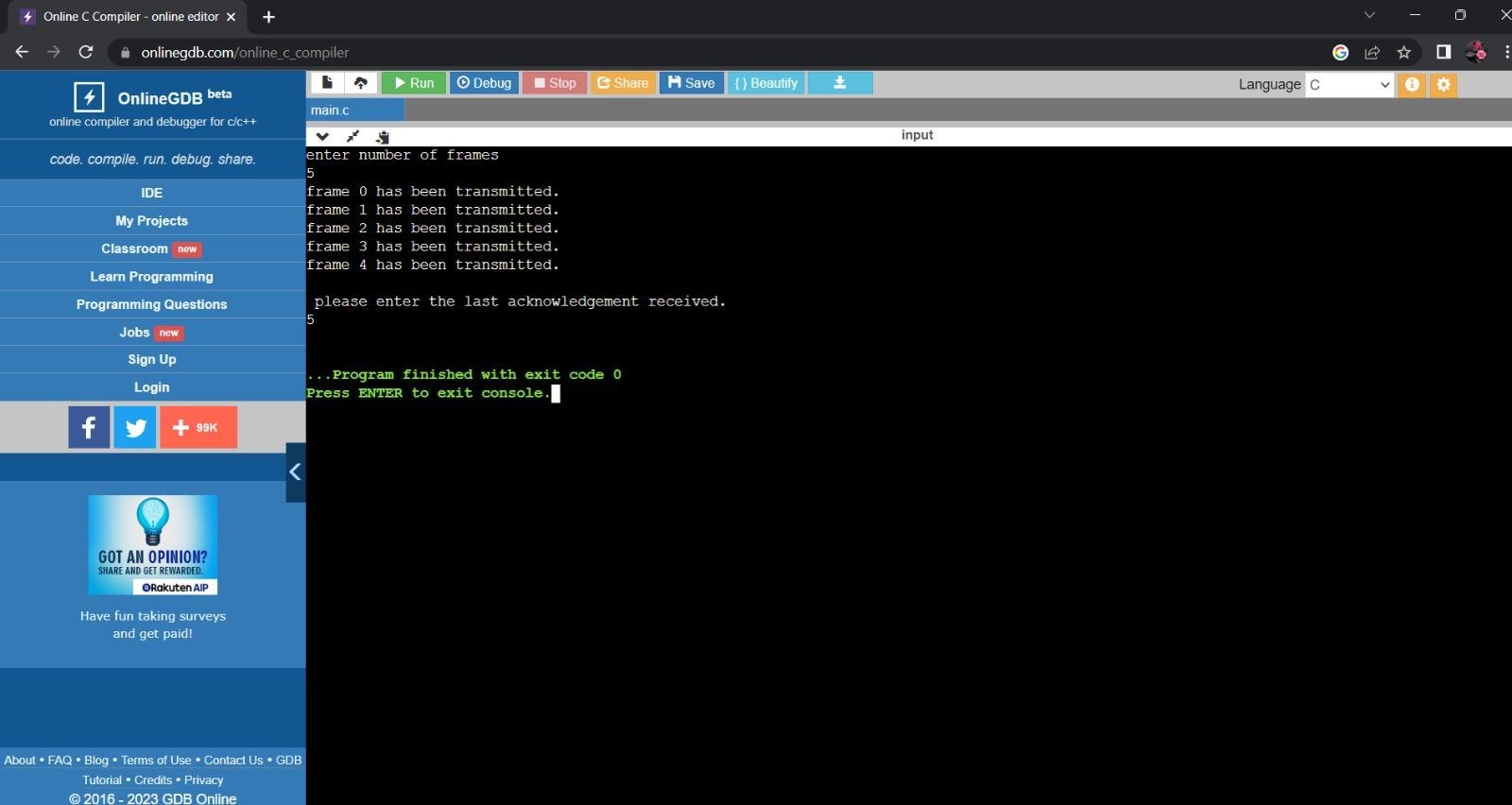
# printf(asctime(ptr));

# printf("Received frame %d\n",seq);

# }

# }

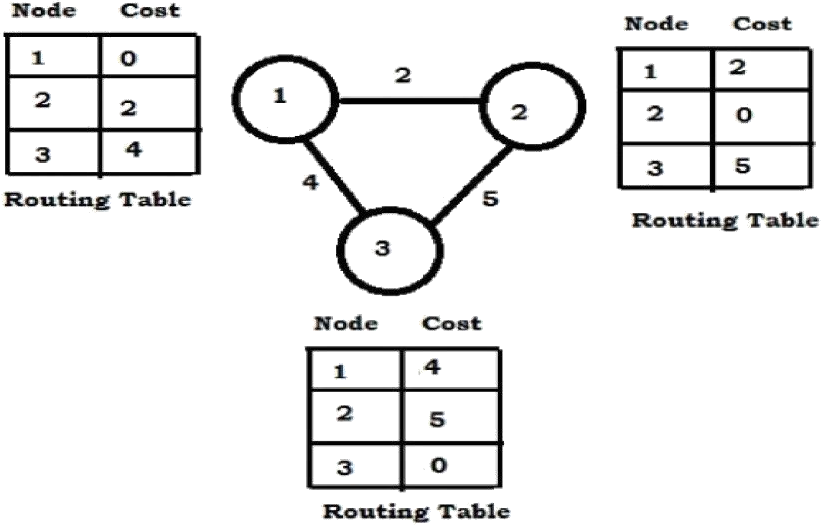
Output:



**Task 6: Implementation Distance vector algorithm. Description:**

Distance Vector Routing Algorithms calculate a best route to reach a destination basedsolely on distance.

E.g. RIP. RIP calculates the reach ability based on hop count. It’s different from link state algorithms which consider some other factors like bandwidth andother metrics to reach a destination. Distance vector routing algorithms are not preferable for complex networks and take longer to converge.



**Program:** #include<stdio.h> #include<string.h> main()

{

int i,j,x,y,n,d,delay=1000;

int edge[50][50],c[50],cost[50][50]; char ch;

printf("\nEnter the No.of nodes in graph: "); scanf("%d",&n);

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

{

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

{

fflush(stdin);

printf("\nIs there are any edge from %d to %d? ",i+1,j+1);

scanf("%d",&d);

if(d)

{

}

else

{

}

}

}

edge[i][j]=1;

edge[i][j]=0;

printf("\nWhich Routing table do u want to find the cost of which destination? "); scanf("%d%d",&x,&y);

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

{

if((edge[i][x-1])==1||(edge[x-1][i])==1)

{

printf("\nEnter the cost of %d node to its neighbour %d:",x,i+1); scanf("%d",&c[i]);

printf("\nEnter the Routing table entry to %d to %d: ",i+1,y); scanf("%d",&cost[i][y-1]);

if(delay>(c[i]+cost[i][y-1]))

{

d=i+1; delay=c[i]+cost[i][y-1];

}

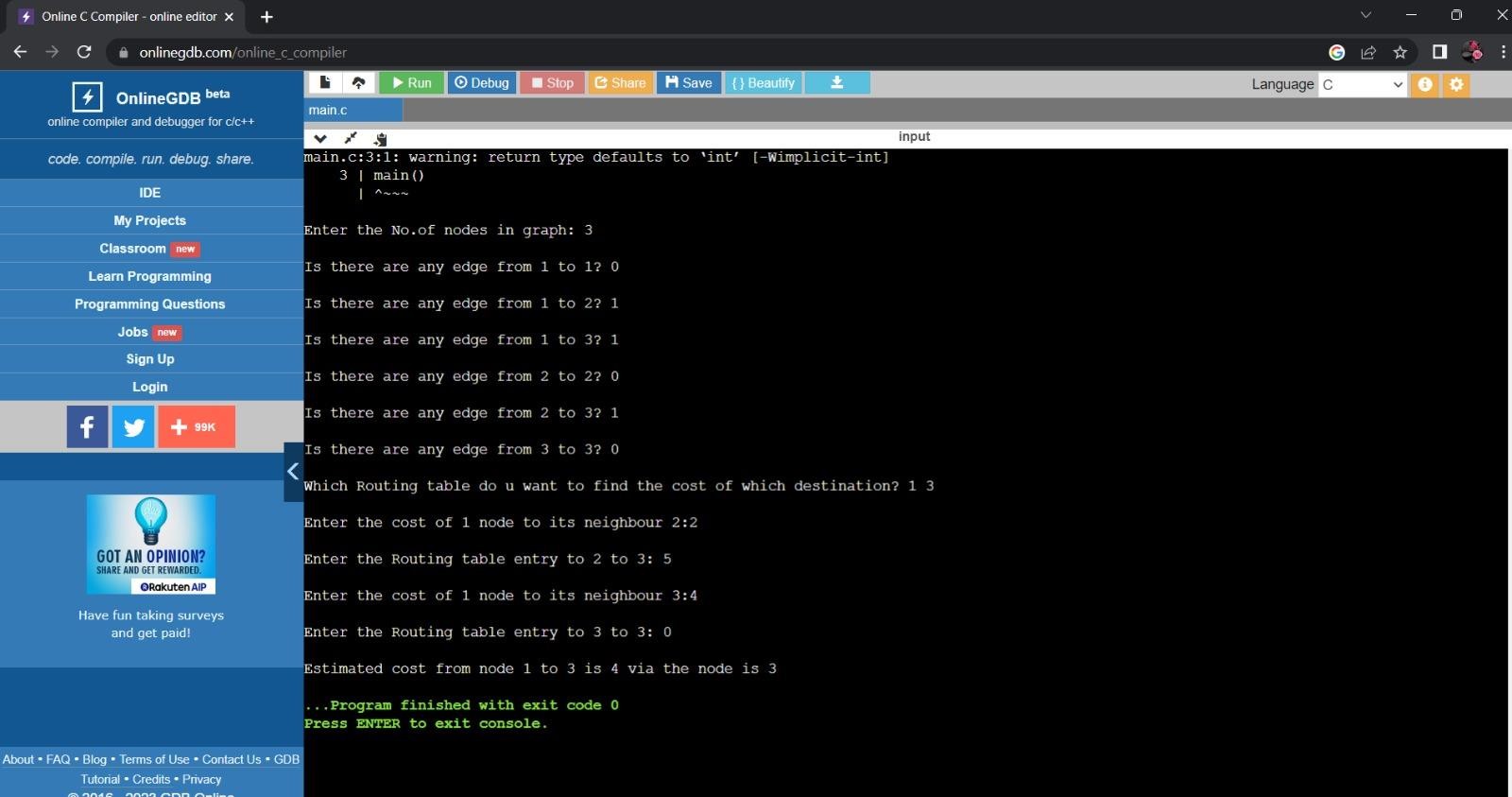
}

}

printf("\nEstimated cost from node %d to %d is %d via the node is %d",x,j,delay,d);

}

Output:



# Task 7: Implementation of Dijkstra’s algorithm. Description:

It is a static routing algorithm. It is used to build a graph of the subnet, with each node of graph representing a router and each arc of the graph representing a communication line. To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph. Different ways of measuring the path length is the number of Hops, Geographical distance in kmts, Mean Queuing delay, Transmission delay, Functions of distance, Bandwidth, Average traffic, communication cost etc.,

This algorithm is given in Fig. 5-8. The global variables n and dist describe the graph andare initialized before shortest\_path is called. The only difference between the programand the algorithm described above is that in Fig. 5-8, we compute the shortest path starting at the terminal node, t, rather than at the source node, s. Since the shortest path from t to s in an undirected graph is the same as the shortest path from s to t, it does not matter at which end we begin (unless there are several shortest paths, in which case reversing the search might discover a different one). The reason for searching backward is that each node is labeled with its predecessor rather than its successor. When the final path is copied into the output variable, path, the path is thus reversed. By reversing the search, the two effects cancel, and the answer is produced in the correct order.

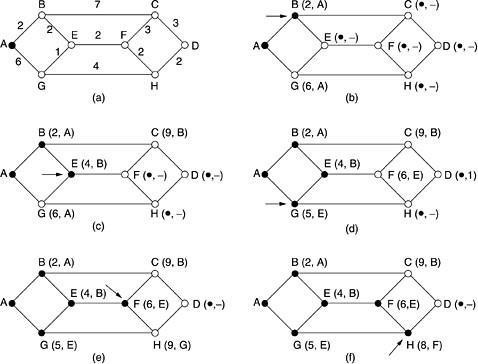


Figure: The first five steps used in computing the shortest path from A to D.The arrows indicate the working Finally, Destination ‘D’ is relabeled as D(10,H).

The path is (D-H-F-E-B-A) as follows:D(10,H) = H(8,F)

= F(6,E)

= E(4,B)

= B(2,A)

= A

**Program:** #include<stdio.h> #include<conio.h> #define MAX\_NODES 50

#define INFINITY 1000 int n;

int shortest\_path(int,int,int a[]); char name[MAX\_NODES];

int dist[MAX\_NODES][MAX\_NODES]; void main()

{

int i,j,l;

char sour,dest,c=65;

int path[MAX\_NODES]; clrscr();

printf("Enter the no. of nodes:"); scanf("%d",&n);

printf("\nEnter the distances between each & every node\n"); printf("if there is no path then enter 0 as its distance\n\n"); printf("\n\t");

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

{

name[i]=c++; printf("%c\t",name[i]);

}

printf("\n\n"); for(i=0;i<n;i++)

{

printf("%c\t",name[i]); for(j=0;j<n;j++)

{

scanf("%d",&dist[i][j]);

}

}

A:

printf("\n Enter the source name:"); sour=getch();

getch(); for(i=0;i<n;i++)

{

if(name[i]==sour)

{

break;

}

}

if(i==n)

{

printf("\n NO node with the given name\n"); goto A;

}

B:

printf("\n\n Enter the distination name:"); dest=getch();

getch(); for(j=0;j<n;j++)

{

if(name[j]==dest)

{

break;

}

if(j==n)

{

}

printf("\n NO node with given destination name\n");

goto B;

}

l=shortest\_path(i,j,path); printf("The shortest path is "); for(i=(l-1);i>0;i--)

{

printf("%c->",name[path[i]]);

}

printf("%c",name[path[i]]); getch();

}

int shortest\_path(int s,int d,int p[])

{

struct state

{

int predecessor; int length;

int label;

}state[MAX\_NODES]; int i,k,min,m; for(i=0;i<n;i++)

{

state[i].predecessor=-1; state[i].length=INFINITY;

state[i].label=0;

}

state[s].length=0; state[s].label=1; k=s;

do

{

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

{

if(dist[k][i]!=0 && state[i].label==0)

{

if(state[k].length+dist[k][i]<state[i].length)

{

}

} k=0;

state[i].predecessor=k; state[i].length=state[k].length+dist[k][i];

}

min=INFINITY;

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

{

if(state[i].label==0 && state[i].length<min)

{

min=state[i].length; k=i;

}

}

state[k].label=1;

}while(k!=d); i=0;

k=d; do

{

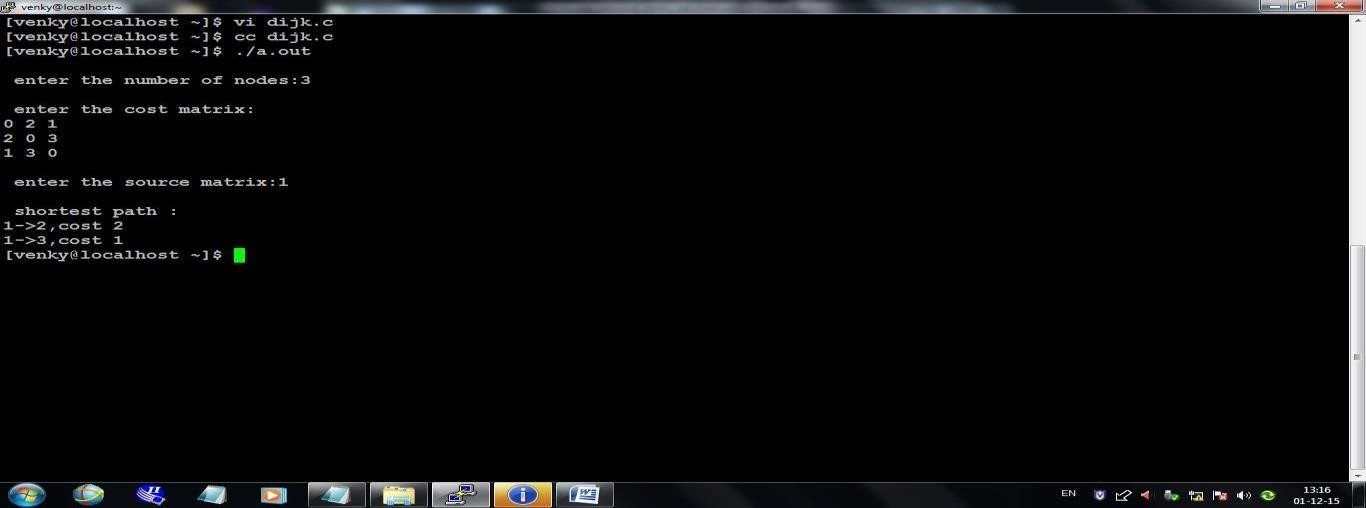
}

Output:

p[i++]=k; k=state[k].predecessor;

}while(k>=0);

printf("\n\n The shortest distance %d \n\n",state[d].length); return(i);



# Task 8: : Implementation of Congestion control using leaky bucket algorithms. Description:

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design,in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.

In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuitsinternally, these methods can be used at the network layer.

Another open loop method to help manage congestion is forcing the packet to betransmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called traffic shaping.

The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one ormore process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flowof packet onto the network, smoothing out bursts and greatly reducing the chances of congestion

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flowof packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.

**Program:** #include<stdio.h> #include<stdlib.h> #define bucketSize 512 void bktInput(int a,int b) {

if(a>bucketSize) printf("\n\t\tBucket overflow");

else {

sleep(2);

while(a>b){

printf("\n\t\t bytes outputted."); a-=b; sleep(2);

}

if (a>0)

printf("\n\t\tLast %d bytes sent\t",a); printf("\n\t\tBucket output successful");

}

}

void main() {

int op,i,pktSize; rand();

printf("Enter output rate : "); scanf("%d",&op); for(i=1;i<=5;i++)

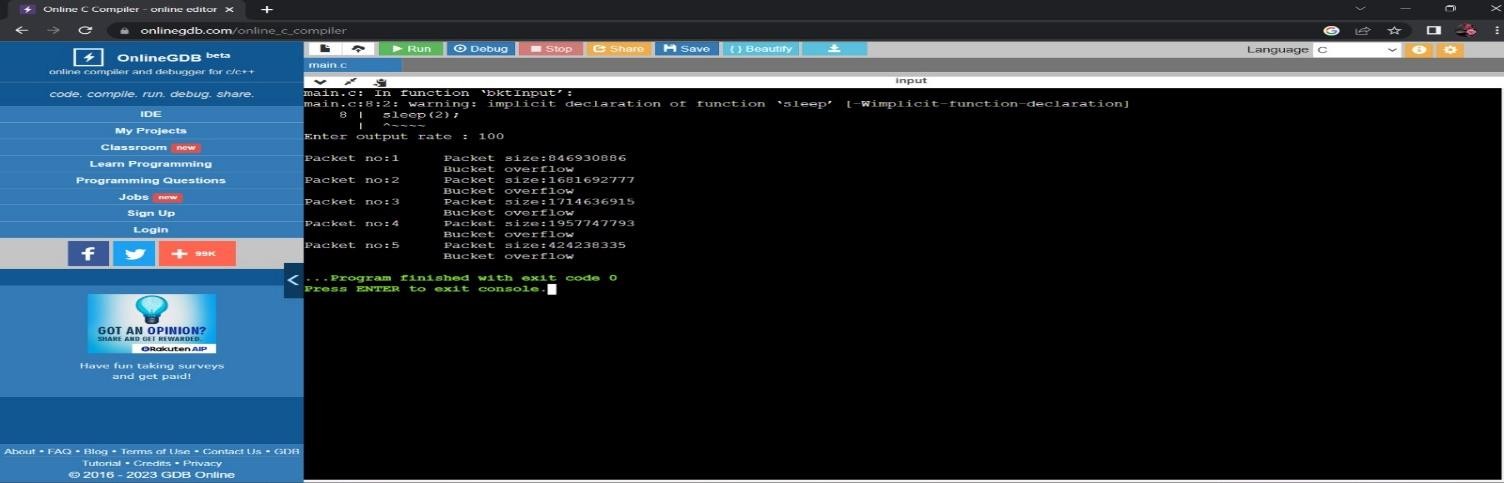
{

sleep(2); pktSize=rand();

printf("\nPacket no:%d \tPacket size:%d ",i,pktSize); bktInput(pktSize,op);

}

}

Output:

# Task 9: Implementation using Socket TCP both client and server. Description:

## What is socket programming?

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

## Stages for server Socket creation:

In Java, a Socket is a part of java.net package and it provides mechanism for the servers and client to communicate over a network. The Socket class is used to create a Socket that is connected to the specific host and port number.

**Ex:** Socket s=new Socket(“localhost”,6666)

## ServerSocket():

In Java, ServerSocket is a class used to create a server socket that wait for requests to come in over the network. It is a part of java.net package and provides a system independent implementation of sever side of a client or server socket connection.

## Some common Constructors:

ServerSocket (): It creates an unbound server socket.

ServerSocket (int port): It creates a ServerSocket, bound to the specified port.

## accept:

The primary method you will use with a ServerSocket is accept (), which listens for a connection to be made to the Socket and accepts it. This method blocks until a connection is made, returning an instance of socket that represents the client socket.

## DataInputStrem:

In Java, DataInputStream is a class that allows an application to read primitive datatypes from underlying input stream in a machine independent way.

## Methods provided by DataInputStream:

readInt (): It reads input bytes and returns an int value.

readUTF (): It reads a string that has been encoded using a modified UTF-8 format. writeUTF (): It is used to write strings in a modified UTF-8 format to an Output Stream. **Stages for Client**

## Socket creation:

In Java, a Socket is a part of java.net package and it provides mechanism for the servers and client to communicate over a network. The Socket class is used to create a Socket that is connected to the specific host and port number.

**Ex:** Socket s=new Socket(“localhost”,6666)

## flush():

In Java, the flush () is used with output stream classes to ensure that all data that has been written to the output stream is actually sent out to the destination.

# Program:

**SERVER PROGRAM:**

import java.io.\*;

import java.net.\*;

public class MyServer {

public static void main(String[] args) {

try {

ServerSocket ss = new ServerSocket(6666);

System.out.println("Server is waiting for a connection...");

Socket s = ss.accept();

System.out.println("Client connected!");// Create input stream to read data from the client

DataInputStream dis = new DataInputStream(s.getInputStream());

String clientMessage = dis.readUTF();

System.out.println("Program Name: " + clientMessage);

String clientMessage1 = dis.readUTF();

System.out.println("Section: " + clientMessage1);

String additionalData = dis.readUTF();

System.out.println("Message Recived: " + additionalData);

dis.close();

s.close();

ss.close();

} catch (Exception e) {

System.out.println(e);

}

}

}**Client Program:**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class MyClient {

public static void main(String[] args) {

try {

Socket s = new Socket("localhost", 6666);

DataOutputStream dout = new DataOutputStream(s.getOutputStream());

dout.writeUTF("Server & Client");

Scanner scanner = new Scanner(System.in);

System.out.print("Enter your section: ");

String userMessage = scanner.nextLine();

dout.writeUTF(userMessage);

dout.flush();

// Read additional parameters (you can modify this part)

String additionalData = "Your additional data here"; // Replace with your actual data

dout.writeUTF(additionalData);

dout.flush();

dout.close();

s.close();

} catch (Exception e) {

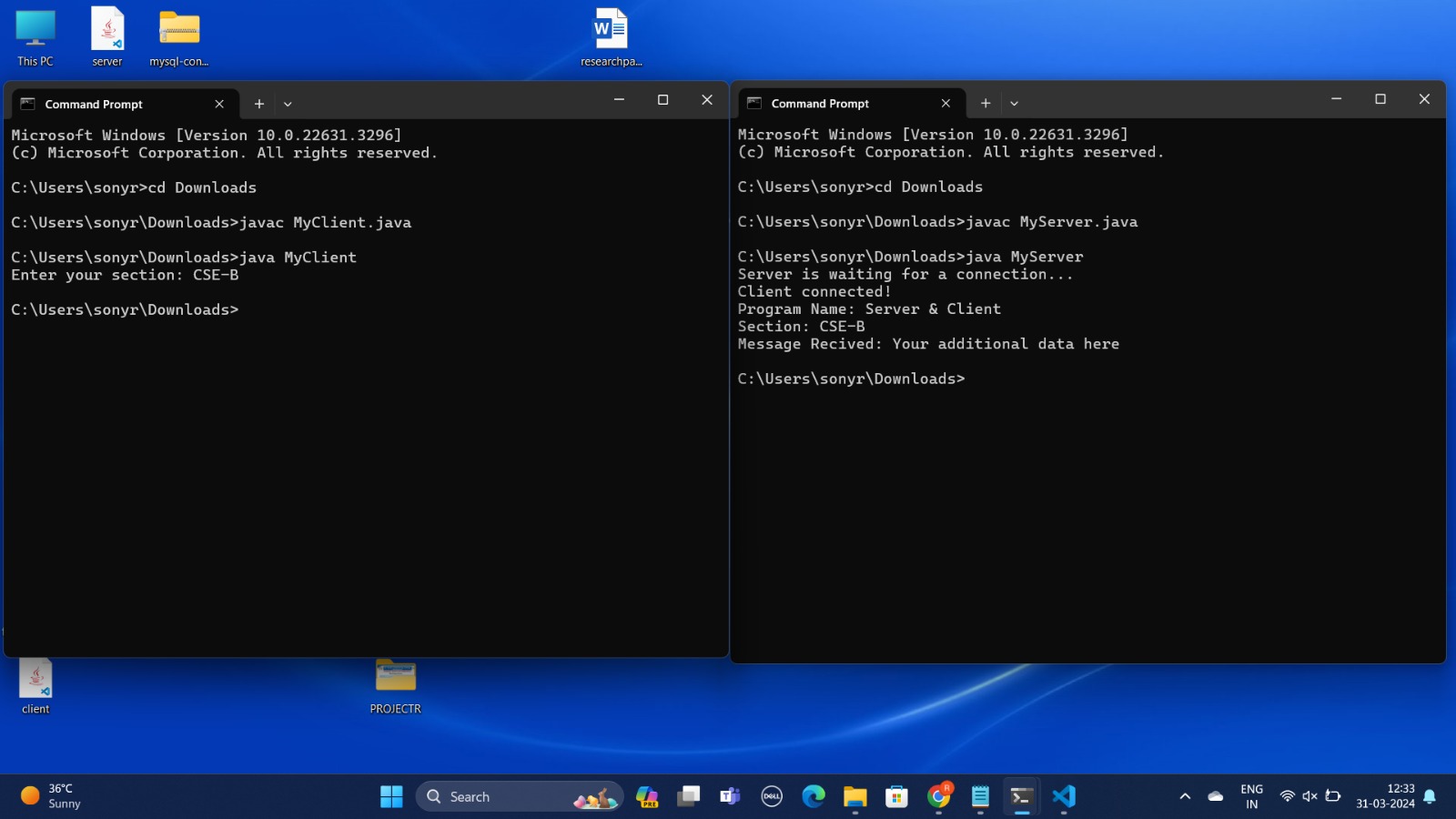
System.out.println(e);

}

}

}

**Output:**



# Task 10: Implementation using Socket UDP both client and server programs. Description:

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.

## The common classes which are used in UDP Client and Server:

**+**

## DatagramSocket:

In Java, the DatagramSocket class represents a connection-less socket for sending and receiving datagram packets using the User Datagram Protocol (UDP).

## Some common constructors:

DatagramSocket (): Creates a datagram socket and binds it to any available port on the local machine. DatagramSocket (int port): Creates and binds the datagram socket to the specified port.

DatagramSocket (int port, InetAddress address): Binds the socket to a specific port and host address.

## Some common Methods:

bind (SocketAddress addr): Binds the socket to a specific address and port. connect (InetAddress address, int port): Connects the socket to a remote address. disconnect (): Disconnects the socket.

send (DatagramPacket p): Sends a datagram packet from the socket. receive (DatagramPacket p): Receives a datagram packet.

## InetAddress:

In Java, the InetAddress class represents an IP address. It provides methods to get the IP of any host name. It Represents both the 32-bit IPv4 address and the 128-bit IPv6 address. An instance of InetAddress consists of an IP address and usually a hostname (depending on whether hostname resolution was performed during creation). **Some common Methods:**

getByName (String host): Returns an instance of InetAddress containing the IP and name of the specified host. getLocalHost (): Returns the instance of InetAddress containing the local host’s name and address. getHostAddress (): Returns the IP address in textual form.

getHostName (): Returns the hostname for this IP address.

## StringBuilder:

In Java, the StringBuilder class is used to create mutable strings. It provides an alternative to the String class, as it creates a mutable sequence of characters.

## some common Constructors:

StringBuilder (): Constructs a string builder with no characters in it and an initial capacity of 16 characters. StringBuilder (String str): Constructs a string builder initialized to the contents of the specified string

## Some common Methods:

append (X x): Appends the string representation of the X type argument to the sequence. delete (int startIndex, int endIndex): Deletes the string from specified startIndex and endIndex. reverse (): Reverses the string2.

charAt (int index): Returns the character at the specified position. length (): Returns the length of the string2.

## DatagramPacket:

In Java, the DatagramPacket class represents a datagram packet, which is used to implement a connectionless packet delivery service. DatagramPacket is a message that can be sent or received.

## Some common Constructors:

DatagramPacket (byte [] buf, int length): Constructs a DatagramPacket for receiving packets of length length. DatagramPacket (byte [] buf, int length, InetAddress address, int port): Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

## Some common Methods:

getAddress(): Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received2.

getData(): Returns the data buffer2.

getLength(): Returns the length of the data to be sent or the length of the data received. getOffset(): Returns the offset of the data to be sent or the offset of the data received. **Program:**

## UDP Client:

// Java program to illustrate Client side

// Implementation using DatagramSocket import java.io.IOException;

import java.net.DatagramPacket; import java.net.DatagramSocket; import java.net.InetAddress; import java.util.Scanner;

public class client {

public static void main (String [] args) throws IOException{ Scanner sc = new Scanner (System.in);

DatagramSocket ds = new DatagramSocket (); InetAddress ip = InetAddress.getLocalHost(); byte buf[] = null;

while (true){

String inp = sc.nextLine(); buf = inp.getBytes();

DatagramPacket DpSend = new DatagramPacket(buf, buf.length, ip, 1234); ds.send(DpSend);

if (inp.equals("bye")) break;

}

}

}

## UDP Server:

// Java program to illustrate Server side

// Implementation using DatagramSocket import java.io.IOException;

import java.net.DatagramPacket; import java.net.DatagramSocket; import java.net.InetAddress; import java.net.SocketException;

public class server

{

public static void main (String [] args) throws IOException { DatagramSocket ds = new DatagramSocket (1234);

byte [] receive = new byte [65535]; DatagramPacket DpReceive = null; while (true) {

DpReceive = new DatagramPacket(receive, receive.length); ds.receive(DpReceive);

System.out.println("Client :-" + data(receive)); if (data(receive).toString().equals("bye")){

System.out.println("Client sent bye. EXITING");

break;

}

receive = new byte [65535];

}

}

// A utility method to convert the byte array

// data into a string representation.

public static StringBuilder data (byte [] a) { if (a == null)

return null;

StringBuilder ret = new StringBuilder (); int i = 0;

while (a[i] != 0){

ret.append((char) a[i]); i++;

}

return ret;

}

}

**Output:**

# 