**LAB RECORD CN CYCLE TEST 2**

**P PREM SAI**

**1BM19CS109**

**1.Write a program for error detecting code using CRC-CCITT (16-bits).**

#include <iostream>

#include <string.h>

using namespace std;

int crc(char \*ip, char \*op, char \*poly, int mode)

{

strcpy(op, ip);

if (mode) {

for (int i = 1; i < strlen(poly); i++) strcat(op, "0");

cout << "modified input" << op <<endl;

}

for (int i = 0; i < strlen(ip); i++) {

if (op[i] == '1') { for (int j = 0; j < strlen(poly); j++) { if (op[i + j] == poly[j])

op[i + j] = '0'; else

op[i + j] = '1';

}

}

}

for (int i = 0; i < strlen(op); i++)

if (op[i] == '1') return 0; return 1;

}

int main()

{

char ip[50], op[50], recv[50]; char

poly[] = "10001000000100001"; int

choice;

cout << "Enter the input message in binary:";

cin >> ip;

cout << "generated polynomial is" << poly <<endl;

crc(ip, op, poly, 1);

cout<<"The checksum is:"<<op+strlen(ip)<<endl;

cout << "The transmitted message is: " << ip << op + strlen(ip) << endl;

cout << "do you want to test error" << endl; cin >> choice; if(choice == 1)

{

int pos,n;

char cp[50]; strcmp(cp, op);

cout<<"Enter the position where to insert error bit"<<endl;

cin>>pos;

cout << "enter bit you wanted to insert" <<endl; cin >> n; cp[pos]=n; if(!strcmp(op, cp))

{

cout << "No error"<<endl;

}

else {

cout << "Error occured"<<endl;

}

return 0;

}

else{ cout << ""<<endl;}

cout << "Enter the recevied message in binary" << endl;

cin >> recv; if (crc(recv, op, poly, 0)) cout << "No error in data" << endl; else

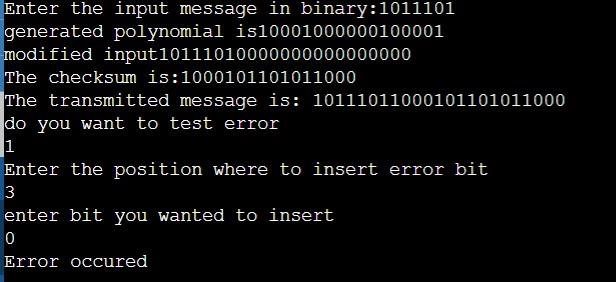
cout << "Error in data transmission has occurred" << endl;

return 0;

}

**OUTPUT :**

Output 1



Output 2



**2. Write a program for distance vector algorithm to find suitable path for transmission.**

#include <bits/stdc++.h> using namespace std; #define MAX 10

int n; class router {

char adj\_new[MAX], adj\_old[MAX]; int

table\_new[MAX], table\_old[MAX];

public: router( ){

for(int i=0;i<MAX;i++) table\_old[i]=table\_new[i]=99;

}

void copy( ){ for(int

i=0;i<n;i++) { adj\_old[i] =adj\_new[i]; table\_old[i]=table\_new[i];

}

}

int equal( ) { for(int

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

if(table\_old[i]!=table\_new[i]||adj\_new[i]!=adj\_old[i])return 0; return

1;

}

void input(int j) { cout<<"Enter 1 if the corresponding router is adjacent to router"

<<(char)('A'+j)<<" else enter 99: "<<endl<<" ";

for(int i=0;i<n;i++) if(i!=j) cout<<(char)('A'+i)<<" "; cout<<"\nEnter matrix:";

for(int i=0;i<n;i++) { if(i==j)

table\_new[i]=0; else

cin>>table\_new[i];

adj\_new[i]= (char)('A'+i);

}

cout<<endl;

}

void display(){

cout<<"\nDestination Router: "; for(int

i=0;i<n;i++) cout<<(char)('A'+i)<<" "; cout<<"\nOutgoing Line: "; for(int i=0;i<n;i++) cout<<adj\_new[i]<<" "; cout<<"\nHop Count: ";

for(int i=0;i<n;i++) cout<<table\_new[i]<<" ";

}

void build(int j) { for(int i=0;i<n;i++) for(int k=0;(i!=j)&&(k<n);k++) if(table\_old[i]!=99) if((table\_new[i]+table\_

new[k])<table\_new[k])

{

table\_new[k]=table\_ne w[i]+table\_new[k];

adj\_new[k]=(char)('A'+i);

}

}

} r[MAX]; void build\_table( ) { int i=0, j=0; while(i!=n) { for(i=j;i<n;i++) { r[i].copy();

r[i].build(i);

}

for(i=0;i<n;i++) if(!r[i].equal())

{ j=i;

break;

}

}

}

int main() { cout<<"Enter the number the routers(<"<<MAX<<"): "; cin>>n;

for(int i=0;i<n;i++) r[i].input(i); build\_table(); for(int i=0;i<n;i++) { cout<<"Router Table entries for router "<<(char)('A'+i)<<":-";

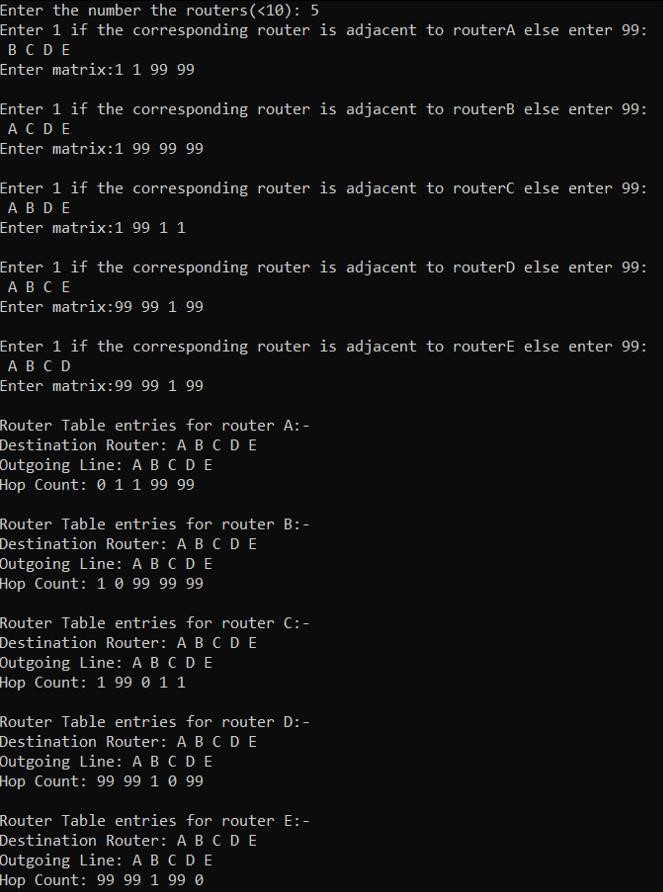
r[i].display();

cout<<endl<<endl;

}

}

**OUTPUT :**



**3. Implement Dijkstra’s algorithm to compute the shortest path for a given topology.**

#include<iostream> #include<climits> using namespace std; int a[30][30],n;

int minimum(int visited[],int dist[])

{

int mindis=10000, mini; for(int i=0;i<n;i++)

{

if(!visited[i] && dist[i]<mindis)

{

mindis=dist[i];

mini=i;

}

}

return mini;

}

void dijkstra(int src)

{

int dist[n],visited[n];

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

{ dist[i]=10000;

visited[i]=0;

}

dist[src]=0;

for(int i=0;i<n-1;i++)

{

int u=minimum(visited,dist); visited[u]=1;

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

{

if(!visited[v] && a[u][v]!=10000 && dist[u]!=10000 &&

(dist[u]+a[u][v])<dist[v])

dist[v]=dist[u]+a[u][v];

}

}

cout<<"Shortest paths to all other vertices from "<<src<<" is "<<endl; cout<<"Vertices\tDistance from source"<<endl; for(int i=0;i<n;i++)

{ if(i!=src) cout<<i<<"\t\t"<<dist[i]<<endl;

}

}

int main()

{

cout<<"Enter the no. of vertices"<<endl;

cin>>n;

cout<<"Enter the weighted adjacency matrix (enter 10000 if there is

no edge)"<<endl; for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++) cin>>a[i][j];

}

int src;

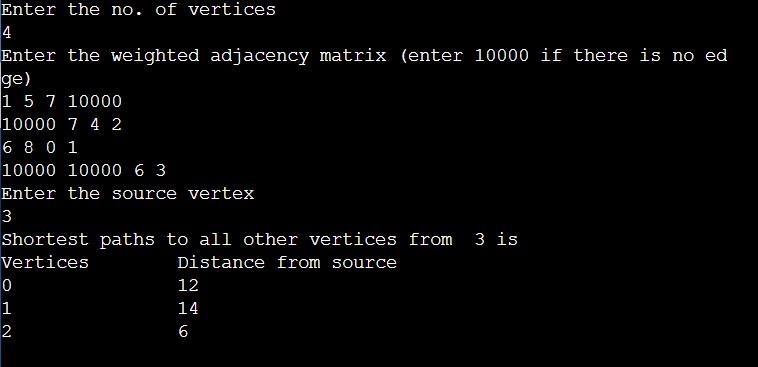
cout<<"Enter the source vertex"<<endl;

cin>>src; dijkstra(src);

return 0;

}

**OUTPUT :**



**4. Write a program for congestion control using Leaky bucket algorithm.**  #include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#define NOF\_PACKETS 5 int main()

{

int packet\_sz[NOF\_PACKETS], i, clk, b\_size, o\_rate, p\_sz\_rm=0, p\_sz, p\_time,

op; for(i = 0; i<NOF\_PACKETS; ++i) packet\_sz[i] = random() % 100; for(i = 0; i<NOF\_PACKETS; ++i)

printf("\npacket[%d]:%d bytes\t", i, packet\_sz[i]);

printf("\nEnter the Output rate:"); scanf("%d", &o\_rate); printf("Enter the Bucket Size:"); scanf("%d", &b\_size); for(i = 0; i<NOF\_PACKETS; ++i)

{

if( (packet\_sz[i] + p\_sz\_rm) > b\_size) if(packet\_sz[i] >

b\_size)/\*compare the packet siz with bucket size\*/ printf("\n\nIncoming packet size (%dbytes) is Greater than bucket capacity

(%dbytes)-PACKET REJECTED", packet\_sz[i], b\_size); else printf("\n\nBucket capacity exceeded-PACKETS REJECTED!!"); else

{

p\_sz\_rm += packet\_sz[i]; printf("\n\nIncoming Packet size: %d", packet\_sz[i]); printf("\nBytes

remaining to Transmit: %d", p\_sz\_rm);

//p\_time = random() \* 10;

//printf("\nTime left for transmission: %d units", p\_time);

//for(clk = 10; clk <= p\_time; clk += 10)

while(p\_sz\_rm>0)

{

sleep(1);

if(p\_sz\_rm)

{

if(p\_sz\_rm <= o\_rate)/\*packet size remaining comparing with

output rate\*/

op = p\_sz\_rm, p\_sz\_rm = 0;

else

op = o\_rate, p\_sz\_rm -= o\_rate; printf("\nPacket of size %d Transmitted", op); printf("----Bytes Remaining to Transmit: %d", p\_sz\_rm); } else

{

printf("\nNo packets to transmit!!");

}

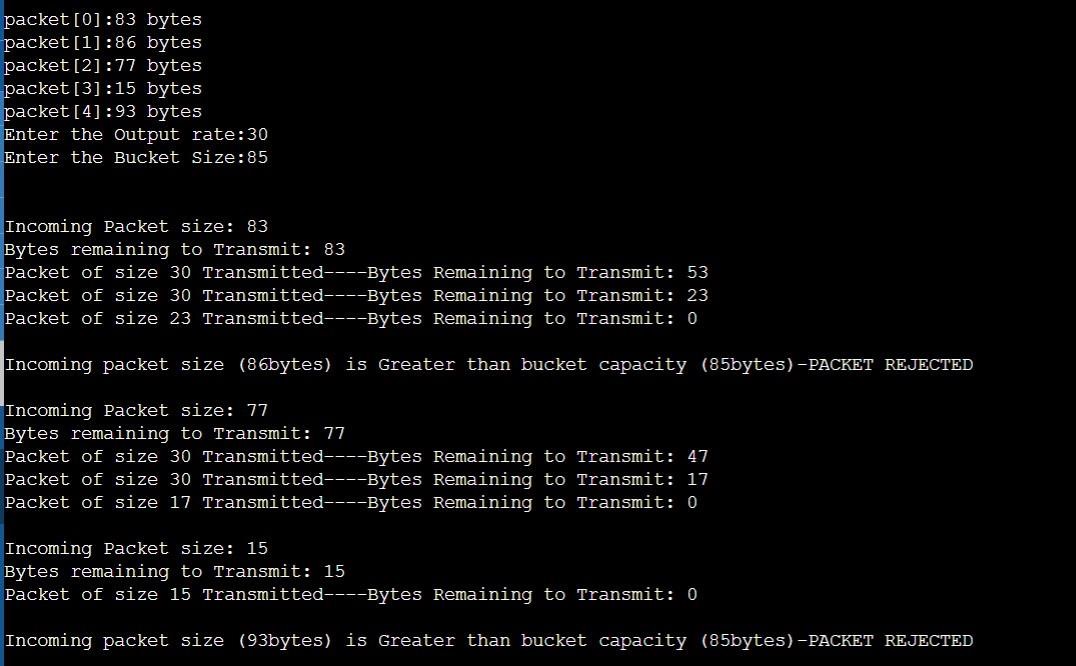
}

}

}

}

**OUTPUT :**



**5. Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.**

ClientTCP.py

from socket import \* serverName = '127.0.0.1'

serverPort = 12000 clientSocket = socket(AF\_INET, SOCK\_STREAM) clientSocket.connect((serverName,serverPort)) sentence = input("\nEnter file name: ")

clientSocket.send(sentence.encode())

filecontents = clientSocket.recv(1024).decode() print ('\nFrom Server:\n') print(filecontents)

clientSocket.close()

ServerTCP.py

from socket import \* serverName="127.0.0.1"

serverPort = 12000 serverSocket = socket(AF\_INET,SOCK\_STREAM) serverSocket.bind((serverName,serverPort))

serverSocket.listen(1) while 1: print ("The server

is ready to receive") connectionSocket, addr = serverSocket.accept() sentence =

connectionSocket.recv(1024).decode()

file=open(sentence,"r")

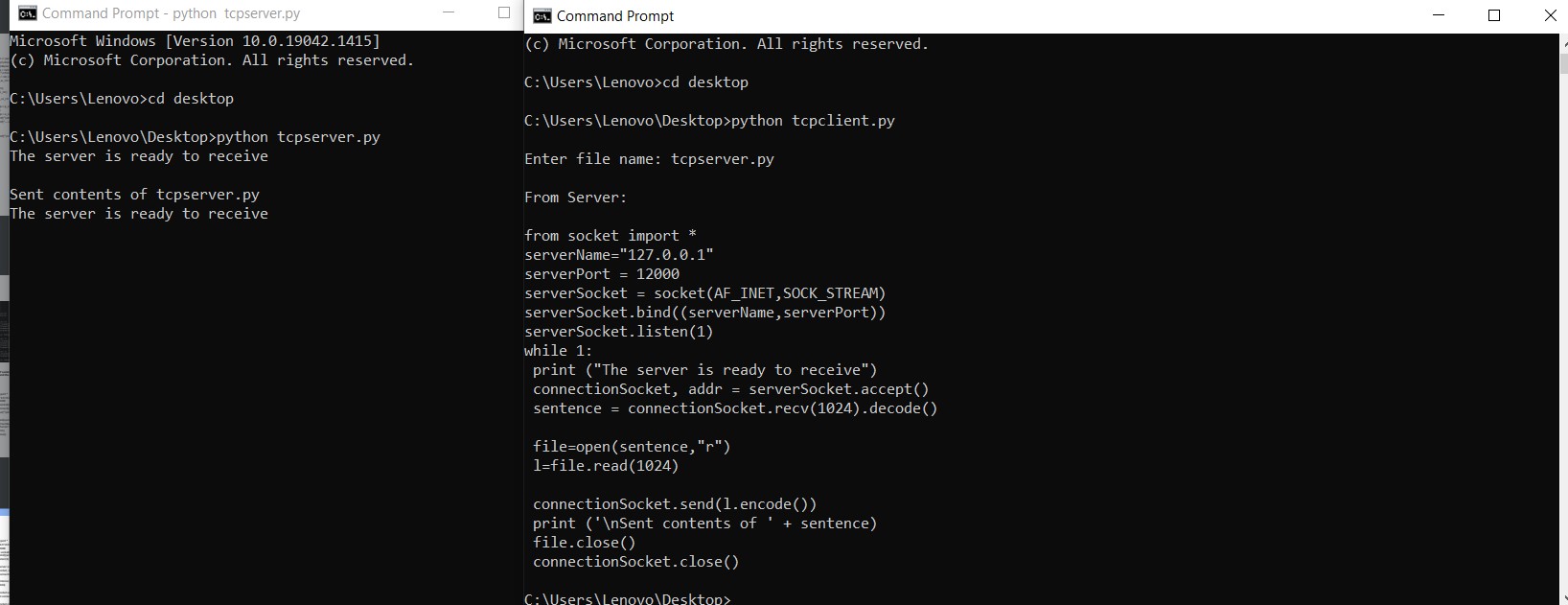
l=file.read(1024)

connectionSocket.send(l.encode())

print ('\nSent contents of ' + sentence) file.close()

connectionSocket.close()

**OUTPUT :**



**6. Using UDP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.**

ClientUDP.py

from socket import \* serverName = "127.0.0.1" serverPort = 12000 clientSocket = socket(AF\_INET, SOCK\_DGRAM)

sentence = input("\nEnter file name: ")

clientSocket.sendto(bytes(sentence,"utf-8"),(serverName, serverPort))

filecontents,serverAddress = clientSocket.recvfrom(2048) print ('\nReply from Server:\n') print

(filecontents.decode("utf-8")) # for i in filecontents: # print(str(i), end = '') clientSocket.close() clientSocket.close()

ServerUDP.py

from socket import \* serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_DGRAM)

serverSocket.bind(("127.0.0.1", serverPort)) print ("The server is ready to receive") while 1:

sentence, clientAddress = serverSocket.recvfrom(2048)

sentence = sentence.decode("utf-8") file=open(sentence,"r")

l=file.read(2048)

serverSocket.sendto(bytes(l,"utf-8"),clientAddress)

print ('\nSent contents of ', end = ' ')

print (sentence) # for i in sentence:

# print (str(i), end = '') file.close()

**OUTPUT :**

