NAME : PRAVEEN KUMAR S

USN : 1BM20CS413

SECTION : C

SUBJECT: COMPUTER NETWORKS

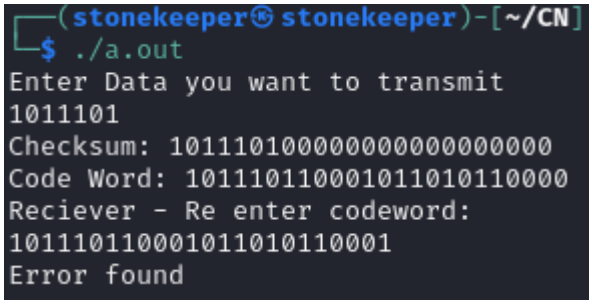
Write a program for error detecting code using CRC-CCITT (16-bits).

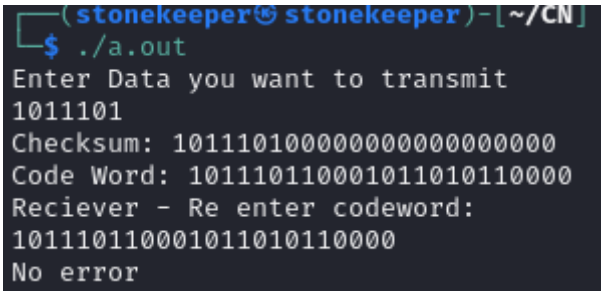
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char g[]="10001000000100001",p[50],c[50];
int n1,n2;

void xor()
{
	int i;
	for(i=0;i<n1;i++)
	{
		if(c[i]==g[i])
		{
			c[i]='0';
		}
		else
		{
			c[i]='1';
		}
	}
}

void checksum()
{
	int i,count;
	for(count=0;count<n1;count++)
	{
		c[count]=p[count];
```

```
        }
        do
        {
                if(c[0]=='1')
                {
                        xor();
                }
                for(i=0;i<n1-1;i++)
                {
                        c[i]=c[i+1];
                }
                c[i]=p[count++];
         }while(count<n2+n1);
}

void main()
{
        int i;
        printf("Enter Data you want to transmit\n");
        scanf("%s",p);
        n1=strlen(g);
        n2=strlen(p);
        for(i=n2;i<n2+n1;i++)
        {
                p[i]='0';
        }
        checksum();
        printf("Checksum: %s\n",p);

        for(i=n2;i<n2+n1;i++)
        {
                p[i]=c[i-n2];
        }
        printf("Code Word: %s\n",p);
```

```
        printf("Reciever - Re enter codeword:\n");
        scanf("%s",p);
        checksum();
        for(i=0;i<strlen(c);i++)
        {
                if(c[i]=='1')
                {
                        printf("Error found\n");
                        exit(0);
                }
        }
        printf("No error\n");
}
```

OUTPUT :





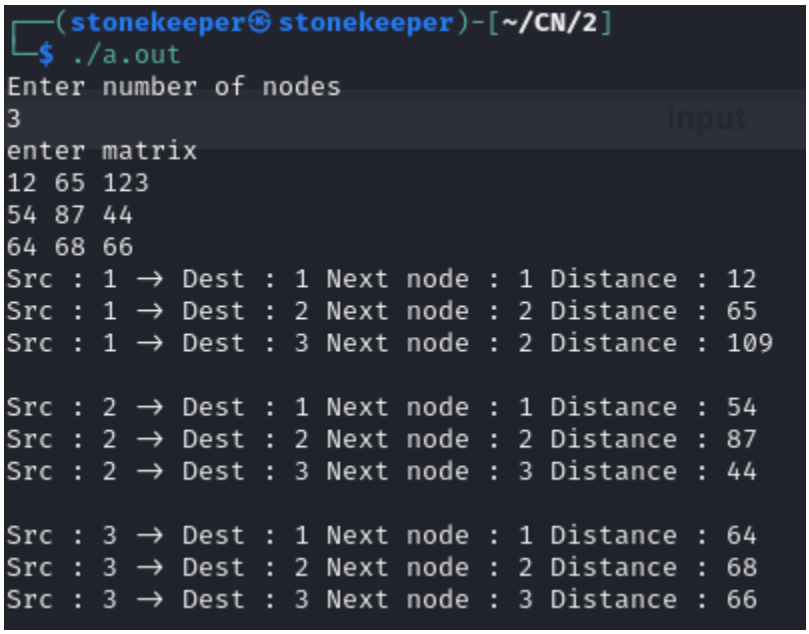
Write a program for distance vector algorithm to find suitable path for transmission.

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
        int dist[15];
        int from[15];
};

void main()
{
        int a[15][15],n=0,i,j,k,count;
```

```
struct node s[10];
printf("Enter number of nodes\n");
scanf("%d",&n);
printf("enter matrix\n");
for(i=1;i<=n;i++)
{
	for(j=1;j<=n;j++)
	{
		scanf("%d",&a[i][j]);
		s[i].dist[j]=a[i][j];//read it as from i to j distance is a[i][j]
		s[i].from[j]=j; // read it as from i to j next node is j
	}
}
//remember floyd's algorithm? all pairs shortest path, apply the same logic!
do
{
	count=0;
	for(k=1;k<=n;k++)
	{
		for(i=1;i<=n;i++)
		{
			for(j=1;j<=n;j++)
			{
				if(s[i].dist[j] > a[i][k] + s[k].dist[j])
				{
					s[i].dist[j] = a[i][k] + s[k].dist[j];
					s[i].from[j]=k;
					count++;
				}
			}
		}
	}
}while(count!=0);
for(i=1;i<=n;i++)
```

```
        {
                for(j=1;j<=n;j++)
                {
                        printf("Src : %d -> Dest : %d Next node : %d Distance :
%d\n",i,j,s[i].from[j],s[i].dist[j]);
                }
                printf("\n");
        }
}
```

OUTPUT :



Implement Dijkstra's algorithm to compute the shortest path for a given topology.

```
#include<bits/stdc++.h>

using namespace std;


#define V 9


int minDistance(int dist[], bool sptSet[])

{


    int min = 9999, min_index;
```

```
    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

void printPath(int parent[], int j)
{
    if (parent[j] == - 1)
        return;

    printPath(parent, parent[j]);

    cout<<j<<" ";
}

void printSolution(int dist[], int n, int parent[])
{
    int src = 0;
    cout<<"Vertex\t Distance\tPath"<<endl;
    for (int i = 1; i < V; i++)
    {
        cout<<"\n"<<src<<" -> "<<i<<" \t \t"<<dist[i]<<"\t\t"<<src<<" ";
        printPath(parent, i);
    }
}

void dijkstra(int graph[V][V], int src)
{

    int dist[V];

    bool sptSet[V];
```

```
    int parent[V];

    for (int i = 0; i < V; i++)
    {
        parent[0] = -1;
        dist[i] = 9999;
        sptSet[i] = false;
    }

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++)
    {
        int u = minDistance(dist, sptSet);

        sptSet[u] = true;

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

            if (!sptSet[v] && graph[u][v] &&
                dist[u] + graph[u][v] < dist[v])
            {
                parent[v] = u;
                dist[v] = dist[u] + graph[u][v];
            }
    }

    printSolution(dist, V, parent);
}

int main()
{
    int graph[V][V];
```

```
    cout<<"Please Enter The Graph (!!! Use 99 for infinity): "<<endl;
    for(int i = 0; i<V; i++)
    {
        for(int j = 0; j<V; j++)
            cin>>graph[i][j];
    }
    cout<<"Enter the source vertex: "<<endl;
    int src;
    cin>>src;

    dijkstra(graph, src);
    cout<<endl;
    return 0;
}
```

OUTPUT:

```
Please Enter The Graph (!!! Use 99 for infinity):
0 4 99 99 99 99 99 8 99
4 0 8 99 99 99 99 11 99
99 8 0 77 99 4 99 99 2
99 99 7 0 9 14 99 99 99
99 99 99 9 0 10 9 99 99
99 99 4 99 10 0 2 99 99
99 99 99 14 99 2 0 1 6
8 11 99 99 99 99 1 0 7
99 99 2 99 99 99 6 7 0
Enter the source vertex:
0
Vertex   Distance       Path

0 -> 1          4               0 1
0 -> 2          12              0 1 2
0 -> 3          23              0 7 6 3
0 -> 4          21              0 7 6 5 4
0 -> 5          11              0 7 6 5
0 -> 6          9               0 7 6
0 -> 7          8               0 7
0 -> 8          14              0 1 2 8
```

Write a program for congestion control using Leaky bucket algorithm.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
```

```
#define NOF_PACKETS 5
/*
int rand (int a)
{
   int rn = (random() % 10) % a;
   return  rn == 0 ? 1 : rn;
}
*/
/*
 #include <stdlib.h>
      long int random(void);
The random() function uses a nonlinear additive feedback random number generator employing a default ta-
      ble of size 31 long integers to return successive pseudo-random numbers in the range from 0 to RAND_MAX.
      The period of this random number generator is very large, approximately 16 * ((2^31) - 1).
*/
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:

```
┌──(stonekeeper㉿stonekeeper)-[~/CN/4]
└─$ ./a.out

packet[0]:83 bytes
packet[1]:86 bytes
packet[2]:77 bytes
packet[3]:15 bytes
packet[4]:93 bytes
Enter the Output rate:30
Enter the Bucket Size:85


Incoming Packet size: 83
Bytes remaining to Transmit: 83
Packet of size 30 Transmitted----Bytes Remaining to Transmit: 53
Packet of size 30 Transmitted----Bytes Remaining to Transmit: 23
Packet of size 23 Transmitted----Bytes Remaining to Transmit: 0

Incoming packet size (86bytes) is Greater than bucket capacity (85bytes)-PACKET REJECTED

Incoming Packet size: 77
Bytes remaining to Transmit: 77
Packet of size 30 Transmitted----Bytes Remaining to Transmit: 47
Packet of size 30 Transmitted----Bytes Remaining to Transmit: 17
Packet of size 17 Transmitted----Bytes Remaining to Transmit: 0

Incoming Packet size: 15
Bytes remaining to Transmit: 15
Packet of size 15 Transmitted----Bytes Remaining to Transmit: 0

Incoming packet size (93bytes) is Greater than bucket capacity (85bytes)-PACKET REJECTED
```

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.

ServerTCP.py

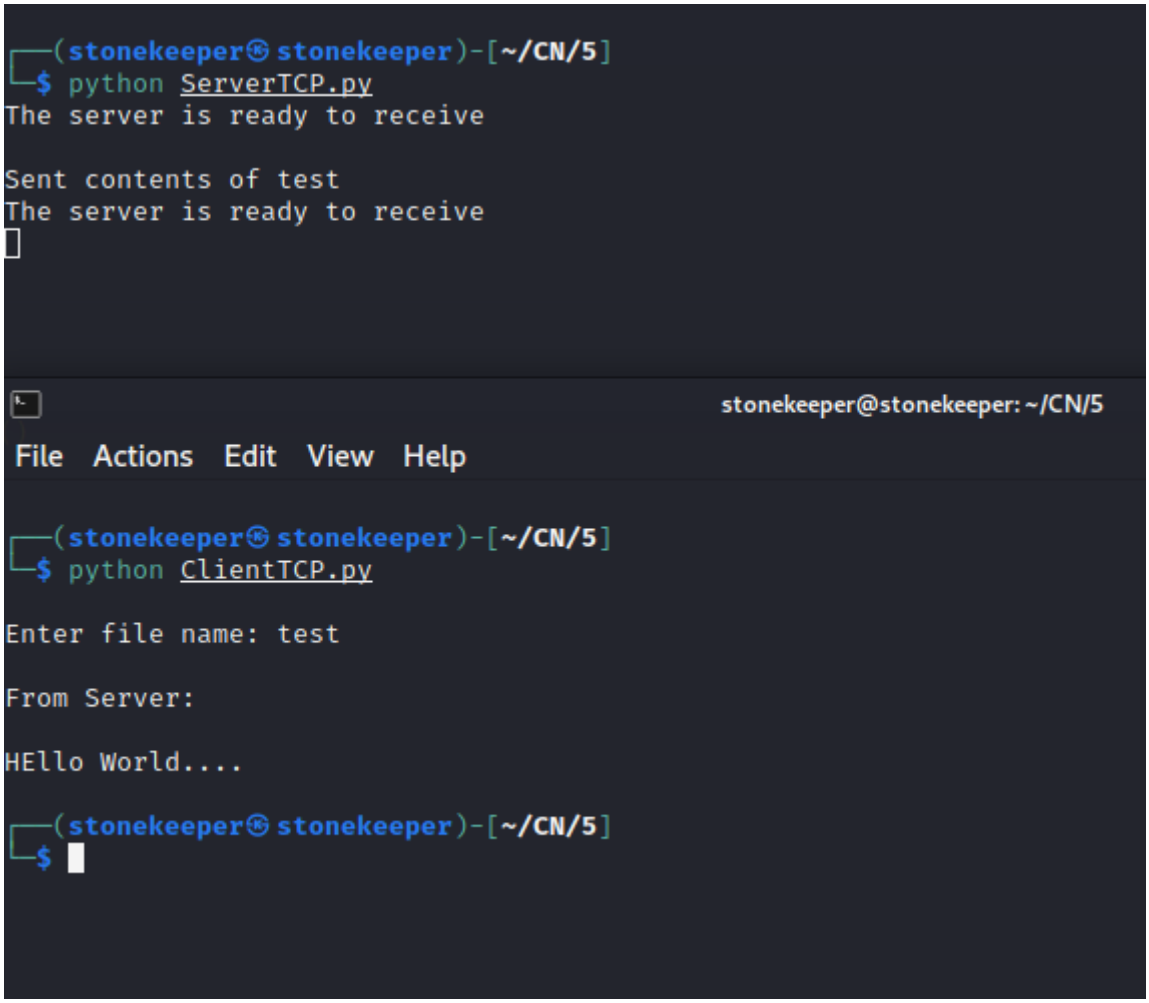
```
from socket import *
serverName="127.0.0.1"
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((serverName,serverPort))
serverSocket.listen(1)
while 1:
    print ("The server is ready to receive")
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()

    file=open(sentence,"r")
    l=file.read(1024)

    connectionSocket.send(l.encode())
    print ('\nSent contents of ' + sentence)
    file.close()
    connectionSocket.close()
```

ClientTCP.py

```
from socket import *
serverName = '127.0.0.1'
```

```
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input("\nEnter file name: ")

clientSocket.send(sentence.encode())
filecontents = clientSocket.recv(1024).decode()
print ('\nFrom Server:\n')
print(filecontents)
clientSocket.close()
```

OUTPUT:



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.

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),clientAddress)
    print ('\nSent contents of ')
    print (sentence)
    file.close()
```

ClientUDP.py

```
from socket import*
serverName = "127.0.0.1"
serverPort = 12000
clientSocket = socket(AF_INET,SOCK_DGRAM)
sentence = raw_input("\nEnter file name: ")
clientSocket.sendto(bytes(sentence),(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 = &#39;&#39;)
clientSocket.close()
clientSocket.close()
```

OUTPUT: