Name- Harshitha Mahadev

USN-1BM19CS065

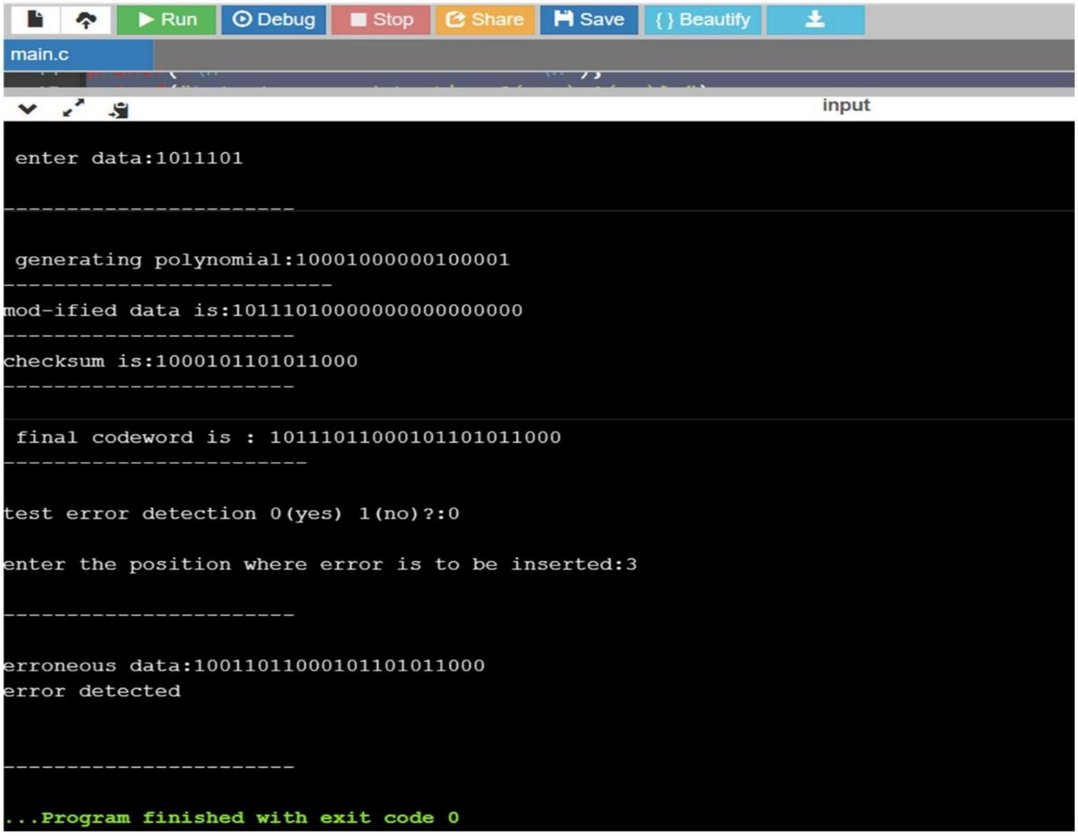
1.Write a program for error detecting code using CRC-CCITT (16-bits)

```
#include <stdio.h>

#include <string.h>

#define N strlen(gen)

char modif[28],checksum[28],gen[28];

int a,e,c,b;

void xor()

{

   for(c=1; c<N; c++)

       checksum[c]=((checksum[c]==gen[c])?'0':'1');

}

void crc()

{

   for(e=0; e<N; e++)

       checksum[e]=modif[e];

   do

   {

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

           xor();

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

           checksum[c]=checksum[c+1];

       checksum[c]=modif[e++];

   }

   while(e<=a+N-1);

}

int main()

{

   int flag=0;

   strcpy(gen,"10001000000100001");
```

```
printf("\n enter data:");
scanf("%s",modif);
printf("\n----------------------\n");
printf("\n generating polynomial:%s",gen);
a=strlen(modif);
for(e=a; e<a+N-1; e++)
   modif[e]='0';
printf("\n-------------------------\n");
printf("mod-ified data is:%s",modif);
printf("\n----------------------\n");
crc();
printf("checksum is:%s",checksum);
for(e=a; e<a+N-1; e++)
   modif[e]=checksum[e-a];
printf("\n----------------------\n");
printf("\n final codeword is : %s",modif);
printf("\n-----------------------\n");
printf("\ntest error detection 0(yes) 1(no)?:");
scanf("%d",&e);
if(e==0)
{
   do
   {
      printf("\nenter the position where error is to be inserted:");
      scanf("%d",&e);
   }
   while(e==0||e>a+N-1);
   modif[e-1]=(modif[e-1]=='0')?'1':'0';
   printf("\n----------------------\n");
   printf("\nerroneous data:%s\n",modif);
}
```

```
    crc();
    for(e=0; (e<N-1)&&(checksum[e]!='1'); e++);
    if(e<N-1)
        printf("error detected\n\n");
    else
        printf("\n no error detected \n\n");
    printf("\n-----------------------");
}
```

OUTPUT:



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

```
class Topology:
    def __init__(self, array_of_points):
        self.nodes = array_of_points
        self.edges = []

    def add_direct_connection(self, p1, p2, cost):
        self.edges.append((p1, p2, cost))
        self.edges.append((p2, p1, cost))

    def distance_vector_routing(self):
        import collections
        for node in self.nodes:
            dist = collections.defaultdict(int)
            next_hop = {node: node}
            for other_node in self.nodes:
                if other_node != node:
                    dist[other_node] = 100000000 # infinity

            # Bellman Ford Algorithm
            for i in range(len(self.nodes)-1):
                for edge in self.edges:
                    src, dest, cost = edge
                    if dist[src] + cost < dist[dest]:
                        dist[dest] = dist[src] + cost
                        if src == node:
                            next_hop[dest] =dest
                        elif src in next_hop:
                            next_hop[dest] = next_hop[src]

            self.print_routing_table(node, dist, next_hop)
            print()

    def print_routing_table(self, node, dist, next_hop):
        print(f'Routing table for {node}:')
        print('Dest \t Cost \t Next Hop')
        for dest, cost in dist.items():
            print(f'{dest} \t {cost} \t {next_hop[dest]}')

array = ['A', 'B', 'C', 'D', 'E']

# Create the network t
= Topology(array)

# Direct connection of each point in the Topology
t.add_direct_connection('A', 'B', 1)
t.add_direct_connection('A', 'C', 5)
t.add_direct_connection('B', 'C', 3)
t.add_direct_connection('B', 'E', 9)
t.add_direct_connection('C', 'D', 4)
t.add_direct_connection('D', 'E', 2)

t.distance_vector_routing()
```

OUTPUT:

Run All

```
Routing table for A:
Dest     Cost    Next Hop
B        1       B
C        4       B
D        8       B
E        10      B
A        0       A

Routing table for B:
Dest     Cost    Next Hop
A        1       A
C        3       C
D        7       C
E        9       E
B        0       B

Routing table for C:
Dest     Cost    Next Hop
A        4       B
B        3       B
D        4       D
E        6       D
C        0       C

Routing table for D:
Dest     Cost    Next Hop
A        8       C
B        7       C
C        4       C
E        2       E
D        0       D

Routing table for E:
Dest     Cost    Next Hop
A        10      B
B        9       B
C        6       D
D        2       D
E        0       E
```

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

```
rom collections import defaultdict

class Graph():
    def __init__(self):
        """
        self.edges is a dict of all possible next nodes
        e.g. {'X': ['A', 'B', 'C', 'E'], ...}
        self.weights has all the weights between two nodes,
        with the two nodes as a tuple as the key
        e.g. {('X', 'A'): 7, ('X', 'B'): 2, ...}
        """
        self.edges = defaultdict(list)
        self.weights = {}

    def addEdge(self, from_node, to_node, weight):
        # Note: assumes edges are bi-directional
        self.edges[from_node].append(to_node)
        self.edges[to_node].append(from_node)
        self.weights[(from_node, to_node)] = weight
        self.weights[(to_node, from_node)] = weight


  def dijsktra(graph, initial, end):
      # shortest paths is a dict of nodes
      # whose value is a tuple of (previous node, weight)
      shortest_paths = {initial: (None, 0)}
      current_node = initial
      visited = set()

      while current_node != end:
          visited.add(current_node)
          destinations = graph.edges[current_node]
          weight_to_current_node = shortest_paths[current_node][1]

          for next_node in destinations:
              weight = graph.weights[(current_node, next_node)] +
  weight_to_current_node
              if next_node not in shortest_paths:
                  shortest_paths[next_node] = (current_node, weight)
              else:
                  current_shortest_weight = shortest_paths[next_node][1]
                  if current_shortest_weight > weight:
                      shortest_paths[next_node] = (current_node, weight)

          next_destinations = {node: shortest_paths[node] for node in
  shortest_paths if node not in visited}
          if not next_destinations:
              return "Route Not Possible"
          # next node is the destination with the lowest weight
          current_node = min(next_destinations, key=lambda k:
  next_destinations[k][1])

      # Work back through destinations in shortest path
      path = []

      while current_node is not None:
          path.append(current_node)
```

```
            next_node=shortest_paths[current_node][0]
            current_node = next_node
      path = path[::-1]
       print('ShortestWeigth:',current_shortest_weight)
       print (path)


       g = Graph()

# Add edges with weight
g.addEdge('a', 'b', 4)
g.addEdge('a', 'c', 2)
g.addEdge('b', 'c', 1)
g.addEdge('b', 'd', 5)
g.addEdge('c', 'd', 8)
g.addEdge('c', 'e', 10)
g.addEdge('d', 'e', 2)
g.addEdge('d', 'z', 6)
g.addEdge('e', 'z', 5)

# Dijkstras Algo dijsktra(g,
'a', 'z')
```

```
Shortest Weigth: 14
['a', 'c', 'b', 'd', 'z']
```

+ Code + Markdown

4. Write a program for congestion control using Leaky bucket algorithm.

```
def
leaky_bucket:

print('-----*****-----')

print(f'The output rate is : {output}') print(f'The bucket size is : {bucket_size} capacity')


packet_no = int(input('Enter number of packets you want to send : '))



for i in range(packet_no):

packet_size = int(input('Enter packet size : '))

if packet_size<bucket_size :

if packet_size <= output:

print(f'Packet number {i} | Packet Size {packet_size} => ')

print('Bucket Output Successful!') print(f'Last {packet_size} bytes sent.') print('-----*****-----')



else:

print(f'Packet number {i} | Packet Size {packet_size} => ')

print('Bucket Output Successful!') print(f'{output} bytes outputted.')


sent = packet_size - output print(f'Last {sent} bytes sent') print('-----*****-----')



else:

print(f'Packet number {i} | Packet Size {packet_size} => ')

print('Bucket *Overflow*') print('-----*****-----')
```

output = int(input('Enter Output Rate : '))

bucket_size = int(input('Enter the bucket size : '))

leaky_bucket(output,bucket_size)

```
PS D:\program files\python> & C:/Python/Python39/python.exe "d:/program files/python/leaky_bucket.py
Enter Output Rate : 100
Enter the bucket size : 500
-----*****-----
The output rate is : 100
The bucket size is : 500 capacity
Enter number of packets you want to send : 5
Enter packet size : 3
Packet number 0 | Packet Size 3 =>
Bucket Output Successful!
Last 3 bytes sent.
-----*****-----
Enter packet size : 33
Packet number 1 | Packet Size 33 =>
Bucket Output Successful!
Last 33 bytes sent.
-----*****-----
Enter packet size : 117
Packet number 2 | Packet Size 117 =>
Bucket Output Successful!
100 bytes outputted.
Last 17 bytes sent
-----*****-----
Enter packet size : 95
Packet number 3 | Packet Size 95 =>
Bucket Output Successful!
Last 95 bytes sent.
-----*****-----
Enter packet size : 949
Packet number 4 | Packet Size 949 =>
Bucket *Overflow*
-----*****-----
```

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.

```
from socket
import *
        serverName = 'DESKTOP-BQNHCT5'
        serverPort = 12001
        clientSocket = socket(AF_INET, SOCK_STREAM)
        clientSocket.connect((serverName,serverPort))
        sentence = input("Enter file name")

        clientSocket.send(sentence.encode())
        filecontents = clientSocket.recv(1024).decode()
        print ('From Server:', filecontents)
        clientSocket.close()
```

```
from
socket
import *


        serverName='DESKTOP-BQNHCT5'
        serverPort = 12001
        serverSocket = socket(AF_INET,SOCK_STREAM)
        serverSocket.bind((serverName,serverPort))
        serverSocket.listen(1)
        print ("The server is ready to receive")
        while 1:
            connectionSocket, addr = serverSocket.accept()
            sentence = connectionSocket.recv(1024).decode()

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

            connectionSocket.send(l.encode())
            file.close()
            connectionSocket.close()
```

```
PS D:\tcp> python -u "d:\tcp\client.py"

Enter file name: server.py

From Server:

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()
```

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.

Client

```
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 = &#39;&#39;)
clientSocket.close()
clientSocket.close()
```

server

```
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 = &#39;&#39;)

file.close()
```

output

```
PS D:\udp> python client.py


Reply from Server:

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 = &#39;&#39;)
    file.close()
PS D:\udp> python client.py
```