

#Write a Program to implement the data link layer framing methods such as

1.Character stuffing

```
def charStuff(flagbyte,escbyte,payload):  
    x = payload.replace(escbyte, escbyte*2)  
    y = x.replace(flagbyte, escbyte+flagbyte)  
    return flagbyte + y + flagbyte
```

```
def charDestuff(flagbyte,escbyte,payload):  
    x = payload.replace(escbyte*2, escbyte)  
    y = x.replace(escbyte+flagbyte, flagbyte)  
    return y[1:-1]
```

```
msg = input('Enter some message : ')
```

```
fb = input('Enter flag byte : ')
```

```
eb = input('Enter Esc byte : ')
```

```
print('Original message : ',msg)
```

```
stf = charStuff(fb,eb,msg)
```

```
print('message after character stuffing : ',stf)
```

```
dstf = charDestuff(fb,eb,stf)
```

```
print('message after character Destuffing : ',dstf)
```

output:

```
Enter some message : hello world  
Enter flag byte : l  
Enter Esc byte : e  
Original message :  hello world  
message after character stuffing :  lheeelelo woreldl  
message after character Destuffing :  hello world
```

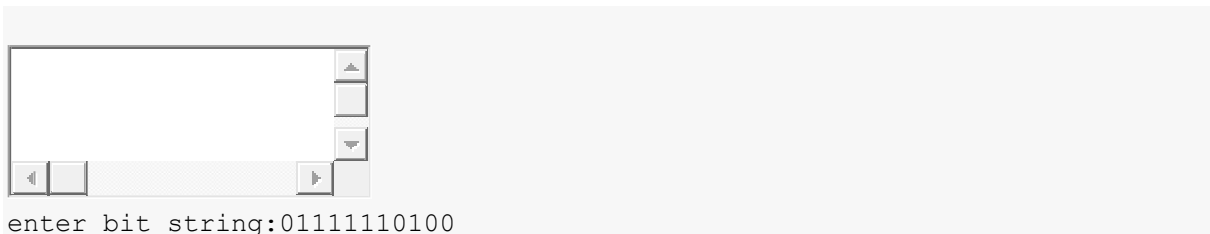
Write a Program to implement the data link layer framing methods such as

ii) bit stuffing.

```
s=input("enter bit string:")
s1=""
c=0
for i in s:
    if i=="1":
        c+=1
    s1+=i
    if c==5:
        s1+="0"
        c=0

print("after bit stuffing the string is:",s1)
```

output:



```
enter bit string:01111110100
after bit stuffing the string is: 011111010100
```

#Write a Program to implement data link layer framing method checksum\

```
def Checksum(x):
    k=8
    c1 = x[0:k]
    c2 = x[k:2*k]
    c3 = x[2*k:3*k]
    c4 = x[3*k:4*k]
    Sum=bin(int(c1,2)+int(c2,2)+int(c3,2)+int(c4,2))
    Sum=Sum[2:]
    print(Sum)
    if(len(Sum) > k):
        x = len(Sum)-k
        Sum = bin(int(Sum[0:x], 2)+int(Sum[x:], 2))[2:]
    if(len(Sum) < k):
        Sum = '0'*(k-len(Sum))+Sum
```

```

Csum = ''
for i in Sum:
    if(i == '1'):
        Csum += '0'
    else:
        Csum += '1'
return Csum
def ChecksumR(x,checksum):
    k=8
    c1 = x[0:k]
    c2 = x[k:2*k]
    c3 = x[2*k:3*k]
    c4 = x[3*k:4*k]
    Sum=bin(int(c1,2)+int(c2,2)+int(c3,2)+int(c4,2)+int(checksum,2))
    Sum=Sum[2:]
    print(Sum)
    if(len(Sum) > k):
        x = len(Sum)-k
        Sum = bin(int(Sum[0:x], 2)+int(Sum[x:], 2))[2:]
    if(len(Sum) < k):
        Sum = '0'*(k-len(Sum))+Sum
    Csum = ''
    for i in Sum:
        if(i == '1'):
            Csum += '0'
        else:
            Csum += '1'
    return Csum

```

```

x=input("enter 32 bit sender string")
y=input("enter 32 bit receiver string")
checksum=Checksum(x)
checksum1=ChecksumR(x,checksum)
print(checksum1) #if checksum==0 true

```

```

output:
enter 32 bit sender string10010101011000111001010011101100
enter 32 bit receiver string10000101011000111001010011101101
1001111000
1011111101
00000000

```

#Write a program for Hamming Code generation for error detection and correction

```
data = [0] * 10
```

```

dataatrec = [0] * 10

print(""Enter 4 bits of data one by one"")
data[0] = int(input())
data[1] = int(input())
data[2] = int(input())
data[4] = int(input())

# Calculation of even parity
data[6] = data[0] ^ data[2] ^ data[4]
data[5] = data[0] ^ data[1] ^ data[4]
data[3] = data[0] ^ data[1] ^ data[2]

print(""\nEncoded data is"")
for i in range(7):
    print(data[i], end="""")

print(""\n\nEnter received data bits one by one"")
for i in range(7):
    dataatrec[i] = int(input())

c1 = dataatrec[6] ^ dataatrec[4] ^ dataatrec[2] ^ dataatrec[0]
c2 = dataatrec[5] ^ dataatrec[4] ^ dataatrec[1] ^ dataatrec[0]
c3 = dataatrec[3] ^ dataatrec[2] ^ dataatrec[1] ^ dataatrec[0]
c = c3 * 4 + c2 * 2 + c1

if c == 0:
    print(""\nNo error while transmission of data"")
else:
    print(""\nError on position", c)
    print(""Data sent :",, end="" "")
    for i in range(7):
        print(data[i], end="""")

    print(""\nData received :",, end="" "")
    for i in range(7):
        print(dataatrec[i], end="""")

    print(""\nCorrect message is"")

# if erroneous bit is 0 we complement it else vice versa
if dataatrec[7-c] == 0:
    dataatrec[7-c] = 1
else:
    dataatrec[7-c] = 0

```

```
for i in range(7):
print(dataatrec[i], end=" ")
```

output:

```
Enter 4 bits of data one by one
1
0
1
0
Encoded data is
1010010
Enter received data bits one by one
1
0
1
0
0
1
0
No error while transmission of data
\
```

#Write a Program to implement on a data set of characters the three CRC polynomials – CRC 12, CRC 16 and CRC CCIP.

```
def crc12(data):
poly = 0b1100000001111
crc = 0
for byte in data:
crc ^= byte << 4
for i in range(8):
if crc & 0x800:
crc = (crc << 1) ^ poly
else:
crc << 1
crc &= 0xFFF
return crc
```

```
def crc16(data):
poly = 0x1021
crc = 0
for byte in data:
crc ^= byte << 8
for i in range(8):
if crc & 0x8000:
crc = (crc << 1) ^ poly
```

```

else:
    crc <<= 1
    crc &= 0xFFFF
    return crc

def crc_ccitt(data):
    poly = 0x1021
    crc = 0xFFFF

    for byte in data:
        crc ^= (byte << 8) & 0xFFFF
        for i in range(8):
            if crc & 0x8000:
                crc = ((crc << 1) ^ poly) & 0xFFFF
            else:
                crc <<= 1
                crc &= 0xFFFF
        return crc

data = b'\x00hello world\x00'
crc12_value = crc12(data)
crc16_value = crc16(data)
crc_ccitt_value = crc_ccitt(data)

print(f'CRC12: {hex(crc12_value)}')
print(f'CRC16: {hex(crc16_value)}')
print(f'CRC CCITT: {hex(crc_ccitt_value)}')

output:
CRC12: 0xc13
CRC16: 0x3be4
CRC CCITT: 0xefeb

```

#Write a Program to implement Stop and Wait Protocol

```
import time
```

```

def send(packet, timeout):
    print(f'Sending packet: {packet}')
    time.sleep(0.5) # Simulating transmission delay

```

```

ack_received = False
start_time = time.time()
while time.time() - start_time < timeout:
    if ack_received:
        break
    try:
        response = input('Enter ACK or NAK: ').strip().lower()
        if response == 'ack':
            ack_received = True
            print('ACK received')
        else:
            print('NAK received, resending packet...')
            time.sleep(0.5) # Simulating transmission delay
            print(f'Sending packet: {packet}')
    except KeyboardInterrupt:
        print('\nSending interrupted by user')
        return

```

```

def receive():
    print('Waiting for packet...')
    time.sleep(0.5) # Simulating transmission delay
    packet_received = False
    while not packet_received:
        try:
            packet = input('Enter packet: ').strip()
            time.sleep(0.5) # Simulating transmission delay
            response = input('Enter ACK or NAK: ').strip().lower()
            if response == 'ack':
                packet_received = True
                print(f'Packet received: {packet}')
                print('Sending ACK...')

```

```

        time.sleep(0.5) # Simulating transmission delay

        print('ACK sent')

    else:

        print('NAK received, waiting for packet...')

except KeyboardInterrupt:

    print('\nReceiving interrupted by user')

    return

```

Example usage:

```
send('hello', 5)
```

```
receive()
```

output:

```

Sending packet: hello
Enter ACK or NAK: nak
NAK received, resending packet...
Sending packet: hello
Waiting for packet...
Enter packet: hello
Enter ACK or NAK: ack
Packet received: hello
Sending ACK...
ACK sent

```

Write a Program to implement Sliding window protocol for Goback N

```
import random
```

```
def transmission(i, N, tf, tt):
```

```
    while i <= tf:
```

```
        z = 0
```

```
        for k in range(i, min(i + N, tf + 1)):
```

```
            print(f"Sending Frame {k}...")
```

```
            tt += 1
```

```
        for k in range(i, min(i + N, tf + 1)):
```

```
            f = random.randint(0, 1)
```

```
            if not f:
```



```

        print(f"Acknowledgment for Frame {k}...")

        z += 1

    else:

        print(f"Timeout!! Frame Number: {k} Not Received")

        print("Retransmitting Window...")

        break

    print()

    i += z

    print(f"Total number of frames which were sent and resent are: {tt}")

```

```
tf = int(input("Enter the Total number of frames: "))
```

```
N = int(input("Enter the Window Size: "))
```

```
i = 1
```

```
tt=0
```

```
transmission(i, N, tf,tt)
```

output:

```

Enter the Total number of frames: 12
Enter the Window Size: 4
Sending Frame 1...
Sending Frame 2...
Sending Frame 3...
Sending Frame 4...
Timeout!! Frame Number: 1 Not Received
Retransmitting Window...

```

```

Total number of frames which were sent and resent are: 4
Sending Frame 1...
Sending Frame 2...
Sending Frame 3...
Sending Frame 4...
Acknowledgment for Frame 1...
Acknowledgment for Frame 2...
Acknowledgment for Frame 3...
Timeout!! Frame Number: 4 Not Received
Retransmitting Window...

```

```

Total number of frames which were sent and resent are: 8
Sending Frame 4...
Sending Frame 5...
Sending Frame 6...
Sending Frame 7...
Acknowledgment for Frame 4...
Acknowledgment for Frame 5...
Timeout!! Frame Number: 6 Not Received

```

Retransmitting Window...

Total number of frames which were sent and resent are: 12

Sending Frame 6...

Sending Frame 7...

Sending Frame 8...

Sending Frame 9...

Acknowledgment for Frame 6...

Timeout!! Frame Number: 7 Not Received

Retransmitting Window...

Total number of frames which were sent and resent are: 16

Sending Frame 7...

Sending Frame 8...

Sending Frame 9...

Sending Frame 10...

Timeout!! Frame Number: 7 Not Received

Retransmitting Window...

Total number of frames which were sent and resent are: 20

Sending Frame 7...

Sending Frame 8...

Sending Frame 9...

Sending Frame 10...

Acknowledgment for Frame 7...

Acknowledgment for Frame 8...

Acknowledgment for Frame 9...

Acknowledgment for Frame 10...

Total number of frames which were sent and resent are: 24

Sending Frame 11...

Sending Frame 12...

Acknowledgment for Frame 11...

Timeout!! Frame Number: 12 Not Received

Retransmitting Window...

Total number of frames which were sent and resent are: 26

Sending Frame 12...

Acknowledgment for Frame 12...

Total number of frames which were sent and resent are: 27

#Write a Program to implement Sliding window protocol for Selective repeat

```
print("Enter window size: ", end="")
```

```
w = int(input())
```

```
print("Enter number of frames to transmit: ", end="")
```

```
f = int(input())
```

```
print(f"\nEnter {f} frames: ", end="")
```

```
frames = list(map(int, input().split()))
```

```
print(frames)
```

```
print("\nWith sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)\n")
```

```
print(f"After sending {w} frames at each stage sender waits for acknowledgement sent by the receiver\n")
```

```
for i in range(0, f):
```

```
    if i % w == 0:
```

```
        print(f'{frames[i]}')
```

```
        print("Acknowledgement of above frames sent is received by sender\n")
```

```
    else:
```

```
        print({frames[i]})
```

```
if f%w!= 0:
```

```
    print("\nAcknowledgement of above frames sent is received by sender")
```

```
output:
```

```
Enter window size: 3
```

```
Enter number of frames to transmit: 5
```

```
Enter 5 frames: 12 4 89 5 6
```

```
[12, 4, 89, 5, 6]
```

```
With sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)
```

```
After sending 3 frames at each stage sender waits for acknowledgement sent by the receiver
```

```
12
```

```
Acknowledgement of above frames sent is received by sender
```

```
{4}
```

```
{89}
```

```
5
```

```
Acknowledgement of above frames sent is received by sender
```

```
{6}
```

Acknowledgement of above frames sent is received by sender

Write a Program to implement Dijkstra's algorithm to compute the Shortest path through a graph

```
import heapq

def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    heap = [(0, start)]
    while heap:
        (distance, current_node) = heapq.heappop(heap)
        if distance > distances[current_node]:
            continue
        for neighbor, weight in graph[current_node].items():
            path_cost = distance + weight
            if path_cost < distances[neighbor]:
                distances[neighbor] = path_cost
                heapq.heappush(heap, (path_cost, neighbor))
    return distances

graph = {
    'A': {'B': 5, 'C': 2},
    'B': {'D': 4, 'E': 2},
    'C': {'B': 8, 'E': 7},
    'D': {'E': 6, 'F': 3},
    'E': {'F': 1},
    'F': {}
}

shortest_paths = dijkstra(graph, 'A')
print(shortest_paths)
final_node=input("destination node")
print(shortest_paths[final_node])
```

output:

```
{'A': 0, 'B': 5, 'C': 2, 'D': 9, 'E': 7, 'F': 8}
destination nodeF
8
```

#Write a Program to implement Distance vector routing algorithm by obtaining routing table at each node (Take an example subnet graph with weights indicating delay between nodes).

```
n = int(input("Enter the number of nodes: "))

dmat = []

rt = []

for i in range(n):

    row = list(map(int, input(f"Enter the cost matrix for node {i+1}: ").split()))

    row[i] = 0

    dmat.append(row)

    rt_node = {'dist': [], 'from': []}

    for j in range(n):

        rt_node['dist'].append(dmat[i][j])

        rt_node['from'].append(j)

    rt.append(rt_node)

count = 0

while True:

    count = 0

    for i in range(n):

        for j in range(n):

            for k in range(n):

                if rt[i]['dist'][j] > dmat[i][k] + rt[k]['dist'][j]:
```

```
rt[i]['dist'][j] = rt[i]['dist'][k] + rt[k]['dist'][j]

rt[i]['from'][j] = k

count += 1
```

```
if count == 0:
```

```
    break
```

```
for i in range(n):
```

```
    print(f"\n\nState value for router {i+1} is:")
```

```
    for j in range(n):
```

```
        print(f"\tnode {j+1} via {rt[i]['from'][j]+1} Distance {rt[i]['dist'][j]}")
```

out put:

```
Enter the number of nodes: 2
Enter the cost matrix for node 1: 1 5
Enter the cost matrix for node 2: 2 4
```

```
State value for router 1 is:
    node 1 via 1 Distance 0
    node 2 via 2 Distance 5
```

```
State value for router 2 is:
    node 1 via 1 Distance 2
    node 2 via 2 Distance 0
```