```
2) def xor(cs, g):
    result = ''
    for i in range(len(cs)):
        result += '0' if cs[i] == g[i] else '1'
    return result
def crc(t, g):
    N = len(g)
    cs = t + '0' * (N - 1)
    e = 0
    while e <= len(t):
        if cs[0] == '1':
            cs = xor(cs, g)
        cs = cs[1:] + t[e]
        e += 1
    return cs
def main():
    options = {
        1: "1100000001111",
        2: "11000000000000101",
        3: "10001000000100001"
    }
    while True:
        print("\n1. CRC12\n2. CRC16\n3. CRC CCIT\n4. Exit")
        b = int(input("Enter your option: "))
        if b == 4:
            break
        elif b not in options:
            print("Invalid option. Please enter again.")
            continue
        g = options[b]
        t = input("Enter data: ")
        print("\nGenerating polynomial:", g)
        cs = crc(t, g)
        print("Checksum:", cs)
        e = int(input("Test error detection (0 - yes, 1 - no)?: "))
        if e == 0:
            error_pos = int(input("Enter the position where error is to be inserted: "))
            if 0 < error_pos <= len(t) + len(g) - 1:
                t = t[:error_pos - 1] + ('0' if t[error_pos - 1] == '1' else '1') + t[error_pos:]
                print("\nErroneous data:", t)
            else:
                print("Invalid position.")
                continue
        cs = crc(t, g)
        if '1' in cs:
            print("Error detected")
        else:
            print("No error detected")

        if int(input("\nEnter 1 to exit, any other number to continue: ")) == 1:
            break
if __name__ == "__main__":
    main()
```

```
3) def main():
    ip = input("Enter input bit sequence: ")
    op = []
    pre_post = ['0', '1', '1', '1', '1', '1', '1', '0']
    # Stuffing
    op.extend(pre_post)
    five = 0
    for bit in ip:
        op.append(bit)
        if bit == '1':
            five += 1
        else:
            five = 0
        if five == 5:
            op.append('0')
            five = 0
    op.extend(pre_post)
    print("\nStuffed Bit Sequence is:", ''.join(op))
    # Destuffing
    decode_op = []
    five = 0
    for i in range(8, len(op) - 8):
        decode_op.append(op[i])
        if op[i] == '1':
            five += 1
        else:
            five = 0
    decode_op = ''.join(decode_op)
    print("Destuffed Bit Sequence is:", decode_op)
if __name__ == "__main__":
    main()

4) def character_stuffing(source):
    char_stuff = ['d', 'l', 'e', 's', 't', 'x']
    j = 6
    i = 0
    while i < len(source):
        if source[i:i + 3] == 'dle':
            char_stuff.extend(['d', 'l', 'e', 'd', 'l', 'e'])
            i += 3
        else:
            char_stuff.append(source[i])
            i += 1
    char_stuff.extend(['d', 'l', 'e', 's', 't', 'x'])
    return ''.join(char_stuff)
def character_destuffing(char_stuff):
    char_destuff = []
    i = 6
    while i < len(char_stuff) - 6:
        if char_stuff[i:i + 3] == 'dle':
            i += 6
        else:
            char_destuff.append(char_stuff[i])
            i += 1
    return ''.join(char_destuff)
def main():
    source = input("Enter plain text: ")
    char_stuff = character_stuffing(source)
    print("After character stuffing:", char_stuff)
    char_destuff = character_destuffing(char_stuff)
    print("After character de-stuffing:", char_destuff)
if __name__ == "__main__":
    main()
```

```
5) import time
def main():
    n = int(input("Enter number of frames: "))
    for i in range(n):
        seq = int(input("Enter sequence: "))
        f = input("Enter frame: ")
        print(time.asctime())  # Simulate receiving frame
        time.sleep(5)  # Simulate processing time
        print(time.asctime())  # Simulate acknowledging frame
        print("Received", seq, "frame")
if __name__ == "__main__":
    main()
```

```
6) MAX_NODES = 50
INFINITY = 1000
def shortest_path(s, d, dist):
    state = [{'predecessor': -1, 'length': INFINITY, 'label': 0} for _ in range(len(dist))]
    state[s]['length'] = 0
    state[s]['label'] = 1
    k = s
    while k != d:
        for i in range(len(dist)):
            if dist[k][i] != 0 and state[i]['label'] == 0:
                if state[k]['length'] + dist[k][i] < state[i]['length']:
                    state[i]['predecessor'] = k
                    state[i]['length'] = state[k]['length'] + dist[k][i]
        k = 0
        minimum = INFINITY
        for i in range(len(dist)):
            if state[i]['label'] == 0 and state[i]['length'] < minimum:
                minimum = state[i]['length']
                k = i
        state[k]['label'] = 1
    path = []
    i = 0
    while k >= 0:
        path.append(k)
        k = state[k]['predecessor']
    return path[::-1], state[d]['length']
def main():
    name = [chr(65 + i) for i in range(MAX_NODES)]
    dist = []
    n = int(input("Enter the number of nodes: "))
    print("\nEnter the distances between each node:")
    print("If there is no path, enter 0 as its distance\n")
    print("\t" + "\t".join(name[:n]))
    for i in range(n):
        print(name[i], end="\t")
        dist.append(list(map(int, input().split())))
    source = input("\nEnter the source name: ")
    destination = input("Enter the destination name: ")
    try:
        s = name.index(source)
        d = name.index(destination)
    except ValueError:
        print("Node with given name not found.")
        return
    path, length = shortest_path(s, d, dist)
    print("\nThe shortest path is:", "->".join([name[node] for node in path]), "\nThe shortest distance:", length)
if __name__ == "__main__":
    main()
```

```
7) def main():
    n = int(input("Enter the number of nodes in the graph: "))
    edge = [[0] * n for _ in range(n)]
    cost = [[0] * n for _ in range(n)]
    for i in range(n):
        for j in range(i, n):
            d = int(input(f"\nIs there an edge from {i + 1} to {j + 1}? (1 for yes, 0 for no): "))
            edge[i][j] = d
            edge[j][i] = d
    x, y = map(int, input("\nEnter the source and destination nodes (separated by space): ").split())
    delay = float('inf')
    for i in range(n):
        if edge[i][x - 1] or edge[x - 1][i]:
            c = int(input(f"\nEnter the cost from node {x} to its neighbor {i + 1}: "))
            cost_entry = int(input(f"Enter the cost from {i + 1} to {y}: "))
            if delay > (c + cost_entry):
                d = i + 1
                delay = c + cost_entry
    print(f"\nEstimated cost from node {x} to {y} is {delay} via node {d}")
if __name__ == "__main__":
    main()
```

```
8) import time
import random
def bucket_input(bucket_size, output_rate):
    if bucket_size > bucketSize:
        print("\n\t\tBucket overflow")
    else:
        time.sleep(2)
        while bucket_size > output_rate:
            print("\n\t\tBytes outputted.")
            bucket_size -= output_rate
            time.sleep(2)
        if bucket_size > 0:
            print(f"\n\t\tLast {bucket_size} bytes sent")
        print("\n\t\tBucket output successful")
def main():
    output_rate = int(input("Enter output rate: "))
    for i in range(1, 6):
        time.sleep(2)
        pkt_size = random.randint(0, 1024)
        print(f"\nPacket no: {i} \tPacket size: {pkt_size}")
        bucket_input(pkt_size, output_rate)
if __name__ == "__main__":
    main()
```

9)

Client:

```
import socket
def main():
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(('localhost', 6666))
        s.sendall(b'Hello Server')
        s.close()
    except Exception as e:
        print(e)
if __name__ == "__main__":
    main()
```

Server:

```
import socket
def main():
    try:
        ss = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        ss.bind(('localhost', 6666))
        ss.listen(1)
        s, _ = ss.accept()
        with s:
            data = s.recv(1024)
            print("message =", data.decode())
    except Exception as e:
        print(e)
if __name__ == "__main__":
    main()
```

10)

Server:

```
import socket
def main():
    try:
        ds = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        ds.bind(('localhost', 1234))
        while True:
            data, addr = ds.recvfrom(65535)
            print("Client:", data.decode())
            if data.decode() == "bye":
                print("Client sent bye.....EXITING")
                break
    except Exception as e:
        print(e)
if __name__ == "__main__":
    main()
```

Client:

```
import socket
def main():
    try:
        ds = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        ip = 'localhost'
        port = 1234
        while True:
            inp = input()
            ds.sendto(inp.encode(), (ip, port))
            if inp == "bye":
                break
    except Exception as e:
        print(e)
if __name__ == "__main__":
    main()
```