Exp.No: 2-CRC

```
def xor():
    for j in range(1, len(gen_poly)):
        check_value[j] = '0' if check_value[j] ==
gen_poly[j] else '1'
def crc():
    global check_value
    check_value = list(data[:len(gen_poly)])
    i = len(gen_poly)
    while i <= data_length + len(gen_poly) - 1:
        if check_value[0] == '1':
            xor()
        check_value = check_value[1:] +
[data[i]] if i < len(data) else check_value[1:]
+ ['0']
        i += 1
def receiver():
    global data
    data = input("\nEnter the received data: ")
    print(f"\nData received: {data}")
    crc()
    if '1' in check_value[:len(gen_poly)-1]:
        print("\nError detected\n")
    else:
        print("\nNo error detected\n")
# Main
data = input("Enter data to be transmitted:
")
gen_poly = input("Enter the Generating
polynomial: ")
data_length = len(data)
# Pad data with zeros
data = list(data + '0' * (len(gen_poly) - 1))
print(f"\nData padded with n-1 zeros:
{''.join(data)}")
# Calculate CRC
crc()
print(f"\nCRC or Check value is:
{''.join(check_value)}")
# Append CRC to data
data = data[:data_length] + check_value
print(f"\nFinal data to be sent:
{''.join(data)}")
# Simulate receiver
receiver()
```

Exp.No: 2-CHECKSUM

```
def sender(arr, n):
    print("\n****SENDER SIDE****")
    total_sum = sum(arr)
    print(f"SUM IS: {total_sum}")
    checksum = ~total_sum & 0xFFFFFFFF  #
1's complement with 32-bit mask to handle
negatives like in C
    print(f"CHECKSUM IS: {checksum}")
    return checksum
def receiver(arr, n, sch):
    print("\n\n****RECEIVER SIDE****")
    total_sum = sum(arr)
    print(f"SUM IS: {total_sum}")
    total_sum += sch
    checksum = ~total_sum & 0xFFFFFFFF  #
1's complement with 32-bit mask
    print(f"CHECKSUM IS: {checksum}")
    if checksum == 0:
        print("No error detected")
    else:
        print("Error detected")
# Main function
n = int(input("ENTER SIZE OF THE STRING:
"))
arr = []
print("ENTER THE ELEMENTS OF THE
ARRAY TO CALCULATE CHECKSUM:")
for _ in range(n):
    arr.append(int(input()))
# Sender side
sch = sender(arr, n)
# Receiver side
receiver(arr, n, sch)
```

Exp.No: 6

TCP Server:

```
import socket
def tcp_server():
    server_socket =
socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server_socket.bind(('localhost', 12345))
    server_socket.listen(1)
    print("TCP server listening...")
    conn, addr = server_socket.accept()
    print(f"Connection from {addr}")
     message = conn.recv(1024).decode()
    print(f"Received: {message}")
     conn.sendall(b"Message received")
    conn.close()
    tcp_server()
```

TCP Client:

```
import socket
def tcp_client():
    client_socket =
socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    client_socket.connect(('localhost', 12345))
      client_socket.sendall(b"Hello, TCP
server!")
    response =
client_socket.recv(1024).decode()
    print(f"Response from server:
{response}")
       client_socket.close()
tcp_client()
```

Exp.No: 6

UDP Server:

```
import socket
def udp_server():
    server_socket =
socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    server_socket.bind(('localhost', 12345))
    print("UDP server listening...")
     while True:
        message, addr =
server_socket.recvfrom(1024)
       print(f"Received from {addr}:
{message.decode()}")
       server_socket.sendto(b"Message
received", addr)
udp_server()
```

UDP Client:

```
import socket
def udp_client():
    client_socket =
socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
     client_socket.sendto(b"Hello, UDP
server!", ('localhost', 12345))
    response, _ =
client_socket.recvfrom(1024)
    print(f"Response from server:
{response.decode()}")
      client_socket.close()
udp_client()
```