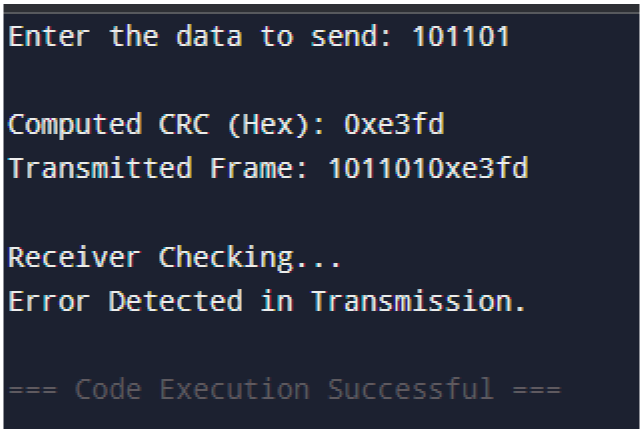
Write a program for congestion control using Leaky bucket algorithm

```
class LeakyBucket:
    def __init__(self, bucket_size, output_rate):
        self.bucket_size = bucket_size
        self.output_rate = output_rate
        self.current_level = 0

    def add_packet(self, packet_size):
        if self.current_level + packet_size > self.bucket_size:
            print(f"Packet of size {packet_size} dropped (Bucket overflow).")
        else:
            self.current_level += packet_size
            print(f"Packet of size {packet_size} added. Current level:
{self.current_level}/{self.bucket_size}")

    def leak(self):
        if self.current_level == 0:
            print("Bucket is empty, nothing to leak.")
            return
        leak_amount = min(self.output_rate, self.current_level)
        self.current_level -= leak_amount
        print(f"Leaked {leak_amount} packets. Remaining: {self.current_level}/{self.bucket_size}")
```

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

```
def crc_ccitt(data, poly=0x1021, init_crc=0xFFFF):
    crc = init_crc
    for byte in data.encode('utf-8'):
        crc ^= (byte << 8)
        for _ in range(8):
            if (crc & 0x8000):
                crc = (crc << 1) ^ poly
            else:
                crc <<= 1
            crc &= 0xFFFF
    return crc
data = input("Enter the data to send: ")
crc_value = crc_ccitt(data)
print(f"\nComputed CRC (Hex): {hex(crc_value)}")
transmitted_data = data + hex(crc_value)
print(f"Transmitted Frame: {transmitted_data}")
received_data = data[:-1] + 'x'
```

```
received_crc = crc_ccitt(received_data)
print("\nReceiver Checking...")
if received_crc == crc_value:
    print("No Error Detected.")
else:
    print("Error Detected in Transmission.")
```



```
Enter the data to send: 101101

Computed CRC (Hex): 0xe3fd
Transmitted Frame: 1011010xe3fd

Receiver Checking...
Error Detected in Transmission.

=== Code Execution Successful ===
```