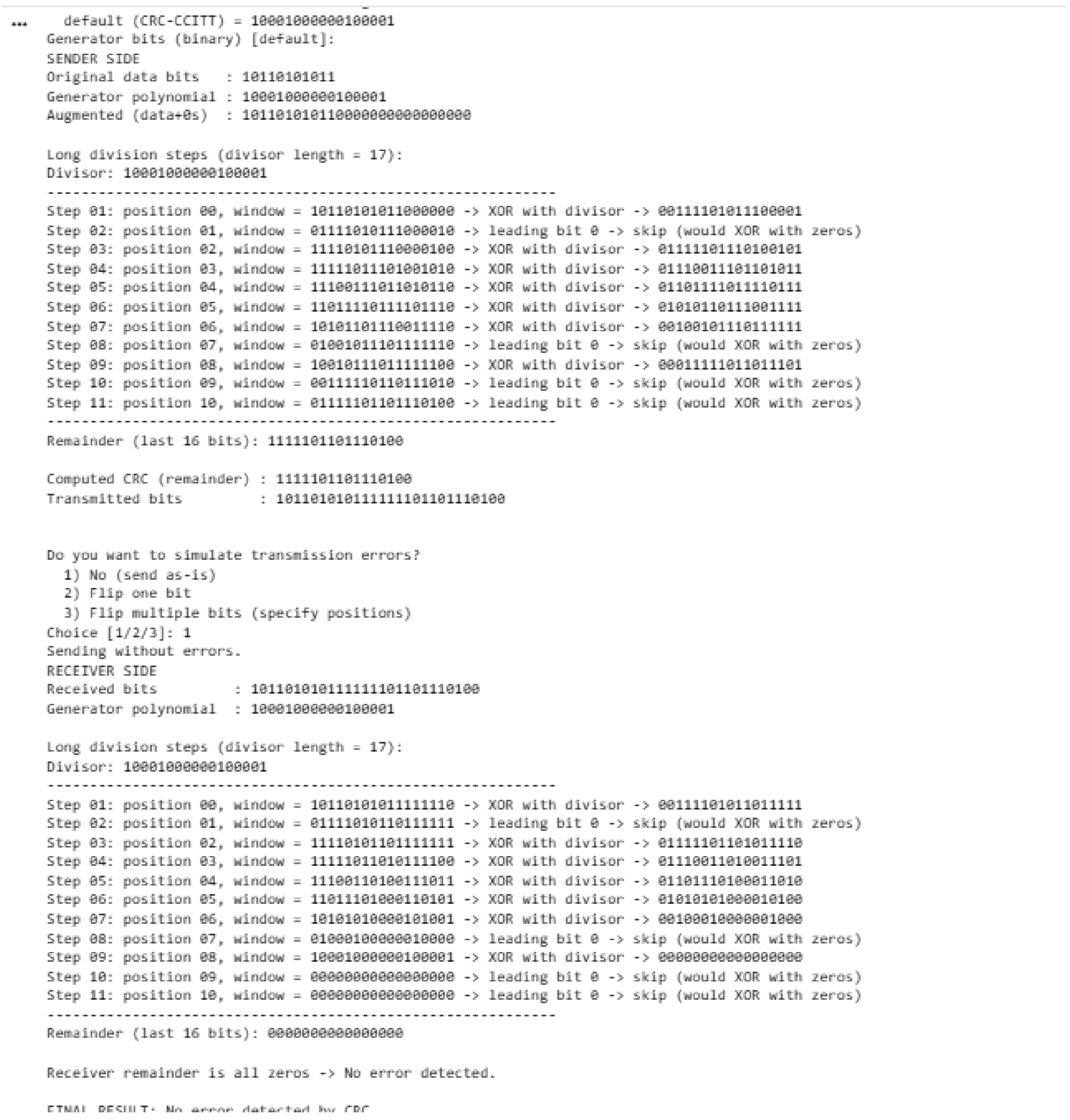
Program – 13:

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

Program:

```
#!/usr/bin/env python3
"""
CRC-CCITT (16-bit) interactive demonstration.
Shows step-by-step binary long division at sender and receiver.
"""

def validate_binary_str(s, name="input"):
    if not s:
        raise ValueError(f"{name} cannot be empty.")
    if any(c not in '01' for c in s):
        raise ValueError(f"{name} must contain only '0' and '1' characters.")
    return s

def xor(a, b):
    """XOR two binary strings of same length, return result string"""
    return ''.join('0' if x == y else '1' for x, y in zip(a, b))

def long_division_show(dividend, divisor, show_steps=True):
    """
    Perform binary long division (modulo-2) and return remainder.
    If show_steps True, prints the division steps.
    dividend/divisor are strings of '0'/'1'. divisor's leading bit must be '1'.
    """
    n = len(divisor)
    # Work on a mutable list for the dividend bits
    work = list(dividend)
    if show_steps:
        print("\nLong division steps (divisor length = {}):".format(n))
        print("Divisor: {}".format(divisor))
        print("-" * 60)

    for i in range(len(dividend) - n + 1):
        # show current window
        window = ''.join(work[i:i+n])
        if show_steps:
            print(f"Step {i+1:02d}: position {i:02d}, window = {window}", end='')
        if window[0] == '1':
            # XOR with divisor
            new_bits = xor(window, divisor)
            if show_steps:
                print(f" -> XOR with divisor -> {new_bits}")
            # write back
            work[i:i+n] = list(new_bits)
        else:
            if show_steps:
                print(" -> leading bit 0 -> skip (would XOR with zeros)")
            # effectively XOR with zeros; we can skip
            # (no change to work)
    remainder = ''.join(work[-(n-1):]) if n > 1 else '0'
    if show_steps:
        print("-" * 60)
        print(f"Remainder (last {n-1} bits): {remainder}\n")
    return remainder

def compute_crc_sender(data_bits, generator_bits, show_steps=True):
    """
    Compute CRC for data_bits using generator_bits.
    Returns the CRC (remainder) and the transmitted bits (data + crc).
    """
    validate_binary_str(data_bits, "Data bits")
    validate_binary_str(generator_bits, "Generator bits")
    if generator_bits[0] != '1':
        raise ValueError("Generator polynomial must start with '1' (highest-order bit).")

    k = len(generator_bits) - 1  # CRC length in bits
    # Append k zeros to data
    augmented = data_bits + '0' * k
    if show_steps:
        print("SENDER SIDE")
        print("Original data bits   : {}".format(data_bits))
        print("Generator polynomial : {}".format(generator_bits))
        print("Augmented (data+0s)  : {}".format(augmented))
    remainder = long_division_show(augmented, generator_bits, show_steps=show_steps)
    # CRC is remainder; transmitted frame is original data + remainder
    transmitted = data_bits + remainder
    if show_steps:
        print(f"Computed CRC (remainder) : {remainder}")
        print(f"Transmitted bits         : {transmitted}\n")
    return remainder, transmitted

def check_crc_receiver(received_bits, generator_bits, show_steps=True):
    """
    Receiver side: check received_bits using generator_bits.
    Returns True if no error detected (remainder all zeros), False otherwise.
    """
    validate_binary_str(received_bits, "Received bits")
    validate_binary_str(generator_bits, "Generator bits")
    if generator_bits[0] != '1':
        raise ValueError("Generator polynomial must start with '1' (highest-order bit).")
    if show_steps:
        print("RECEIVER SIDE")
        print("Received bits         : {}".format(received_bits))
        print("Generator polynomial  : {}".format(generator_bits))
    remainder = long_division_show(received_bits, generator_bits, show_steps=show_steps)
    ok = all(ch == '0' for ch in remainder)
    if show_steps:
        if ok:
            print("Receiver remainder is all zeros -> No error detected.")
        else:
            print("Receiver remainder is NOT all zeros -> Error detected!")
    return ok, remainder

def demo_interactive():
    print("=== CRC-CCITT (16-bit) Demo ===")
    print("Enter data bits (binary), e.g. 11010011101100")
    data_bits = input("Data bits: ").strip()
    # Default CRC-CCITT generator: x^16 + x^12 + x^5 + 1
    default_generator = "10001000000100001"  # 17 bits (degree 16 down to 0)
    print("\nPress Enter to use default CRC-CCITT generator:")
    print(f"  default (CRC-CCITT) = {default_generator}")
    gen_in = input("Generator bits (binary) [default]: ").strip()
    generator_bits = gen_in if gen_in else default_generator

    # Sender computes CRC and shows steps
    try:
        crc, transmitted = compute_crc_sender(data_bits, generator_bits, show_steps=True)
    except ValueError as e:
        print("Input error:", e)
        return

    # Ask if we should flip bits (simulate errors)
    print("\nDo you want to simulate transmission errors?")
    print("  1) No (send as-is)")
    print("  2) Flip one bit")
    print("  3) Flip multiple bits (specify positions)")
    choice = input("Choice [1/2/3]: ").strip() or '1'

    received = transmitted
    if choice == '2':
        pos = input(f"Enter bit position to flip (0..{len(transmitted)-1}), 0 is leftmost: ").strip()
        try:
            p = int(pos)
            if not (0 <= p < len(transmitted)):
                raise ValueError
            b = '1' if transmitted[p] == '0' else '0'
            received = transmitted[:p] + b + transmitted[p+1:]
            print(f"Flipped bit at position {p}. Received bits: {received}")
        except:
            print("Invalid position; no flip performed.")
    elif choice == '3':
        pos_list = input(f"Enter space-separated positions to flip (0..{len(transmitted)-1}): ").strip().split()
        changed = list(transmitted)
        for pos in pos_list:
            try:
                p = int(pos)
                if 0 <= p < len(transmitted):
                    changed[p] = '1' if changed[p] == '0' else '0'
            except:
                pass
        received = ''.join(changed)
        print(f"Received bits after flips: {received}")
    else:
        print("Sending without errors.")

    # Receiver checks CRC
    ok, rem = check_crc_receiver(received, generator_bits, show_steps=True)
    if ok:
        print("\nFINAL RESULT: No error detected by CRC.")
    else:
        print("\nFINAL RESULT: Error detected by CRC.")

if __name__ == "__main__":
    demo_interactive()
```
Output:



```
...   default (CRC-CCITT) = 10001000000100001
    Generator bits (binary) [default]:
    SENDER SIDE
    Original data bits   : 10110101011
    Generator polynomial : 10001000000100001
    Augmented (data+0s)  : 101101010110000000000000000

    Long division steps (divisor length = 17):
    Divisor: 10001000000100001
    ------------------------------------------------------------
    Step 01: position 00, window = 10110101011000000 -> XOR with divisor -> 00111101011100001
    Step 02: position 01, window = 01111010111000010 -> leading bit 0 -> skip (would XOR with zeros)
    Step 03: position 02, window = 11110101110000100 -> XOR with divisor -> 01111101110100101
    Step 04: position 03, window = 11111011101001010 -> XOR with divisor -> 01110011101101011
    Step 05: position 04, window = 11100111011010110 -> XOR with divisor -> 01101111011110111
    Step 06: position 05, window = 11011110111101110 -> XOR with divisor -> 01010110111001111
    Step 07: position 06, window = 10101101110011110 -> XOR with divisor -> 00100101110111111
    Step 08: position 07, window = 01001011101111110 -> leading bit 0 -> skip (would XOR with zeros)
    Step 09: position 08, window = 10010111011111100 -> XOR with divisor -> 00011111011011101
    Step 10: position 09, window = 00111110110111010 -> leading bit 0 -> skip (would XOR with zeros)
    Step 11: position 10, window = 01111101101110100 -> leading bit 0 -> skip (would XOR with zeros)
    ------------------------------------------------------------
    Remainder (last 16 bits): 1111101101110100

    Computed CRC (remainder) : 1111101101110100
    Transmitted bits         : 101101010111111101101110100


    Do you want to simulate transmission errors?
      1) No (send as-is)
      2) Flip one bit
      3) Flip multiple bits (specify positions)
    Choice [1/2/3]: 1
    Sending without errors.
    RECEIVER SIDE
    Received bits         : 101101010111111101101110100
    Generator polynomial  : 10001000000100001

    Long division steps (divisor length = 17):
    Divisor: 10001000000100001
    ------------------------------------------------------------
    Step 01: position 00, window = 10110101011111110 -> XOR with divisor -> 00111101011011111
    Step 02: position 01, window = 01111010110111111 -> leading bit 0 -> skip (would XOR with zeros)
    Step 03: position 02, window = 11110101101111111 -> XOR with divisor -> 01111101101011110
    Step 04: position 03, window = 11111011010111100 -> XOR with divisor -> 01110011010011101
    Step 05: position 04, window = 11100110100111011 -> XOR with divisor -> 01101110100011010
    Step 06: position 05, window = 11011101000110101 -> XOR with divisor -> 01010101000010100
    Step 07: position 06, window = 10101010000101001 -> XOR with divisor -> 00100010000001000
    Step 08: position 07, window = 01000100000010000 -> leading bit 0 -> skip (would XOR with zeros)
    Step 09: position 08, window = 10001000000100001 -> XOR with divisor -> 00000000000000000
    Step 10: position 09, window = 00000000000000000 -> leading bit 0 -> skip (would XOR with zeros)
    Step 11: position 10, window = 00000000000000000 -> leading bit 0 -> skip (would XOR with zeros)
    ------------------------------------------------------------
    Remainder (last 16 bits): 0000000000000000

    Receiver remainder is all zeros -> No error detected.

    FINAL RESULT: No error detected by CRC
```
]