Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology, Pune-37

(An autonomous Institute of Savitribai Phule Pune University)

Department of Computer Engineering

Division	**TY-CS-D**
Batch	**2**
GR-No	**12311330**
Roll No	**8**
Name	**Arnav Satpal Sowale**

Assignment No.3

Title: Write a program for error detection and correction for 7/8 bits ASCII codes using Hamming Codes and CRC. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.

Steps for Error Detection & Correction (Hamming + CRC) with Wireshark

1. **Choose ASCII Data**
    - Take 7/8-bit ASCII characters as input for transmission.

2. **Apply Hamming Code (Error Correction)**
    - Encode each ASCII character into Hamming(12,8).
    - Add parity bits for error detection and single-bit error correction.

3. **Apply CRC (Error Detection)**
    - Append CRC bits (using a generator polynomial) to detect burst errors.

4. **Transmit Data Peer-to-Peer**
    - Use socket programming (TCP/UDP) to send encoded data between two systems (or two terminals on same machine).

5. **Capture Packets in Wireshark**
    - Start Wireshark capture.
    - Use a display filter like ip.addr==<peer_IP> or tcp.port==<port>.
    - Observe the transmitted ASCII + Hamming + CRC encoded packets.

6. **Introduce Transmission Errors**
    - Simulate bit flips in transmitted packets.
    - Observe Hamming correction (1-bit) and CRC detection (multi-bit).

7. **Verify at Receiver**
    - Receiver decodes Hamming, corrects errors.
    - CRC check ensures integrity.
    - Recovered ASCII text should match original.

Program Code(CRC):-

```
import java.util.Scanner;

public class CRC
{

   public static String crcEncode(String data, String divisor)
   {
      int m = divisor.length();        // append (m-1) zeros

      StringBuffer codeword = new StringBuffer(data).append("0".repeat(m - 1));

      // compute remainder using modulo-2 division
      String remainder = divide(codeword.toString(), divisor);

      System.out.println("Bits to Append: " + remainder);

      // replace last (m-1) zeros with remainder
      codeword.replace(codeword.length() - (m - 1), codeword.length(), remainder);

      return codeword.toString();
   }

   // Modulo-2 Division
   private static String divide(String dividend, String divisor)
   {
      char[] result = dividend.toCharArray();
      int m = divisor.length();

      for (int i = 0; i <= dividend.length() - m; i++)
      {
         if (result[i] == '1')
         {
            for (int j = 0; j < m; j++)
            {
               result[i + j] = (result[i + j] == divisor.charAt(j)) ? '0' : '1'; // XOR
            }
         }
      }
```

```
        // remainder is last (m-1) bits

        return new String(result).substring(dividend.length() - (m - 1));
    }


    public static String crcDecode(String data, String divisor)
    {
        String remainder = divide(data, divisor);
            return remainder.contains("1") ? "Error Occurred : Data Mismatch..."    : "Data Properly
Transmitted...";
    }


    public static void main(String[] args)
    {
        Scanner x = new Scanner(System.in);

        System.out.println("ENTER DATA:");
        String data = x.nextLine();

        System.out.println("ENTER DIVISOR:");
        String div = x.nextLine();

        String codeword = crcEncode(data, div);
        System.out.println("Codeword = " + codeword);

        // Receiver test
        System.out.println("ENTER THE MISMATCH OR CORRECT STRING:");
        String received = x.nextLine();

        String status = crcDecode(received, div);
        System.out.println("Receiver Status = " + status);

        x.close();
    }
}
```

Output:-

```
ENTER DATA:
1010
ENTER DIVISOR:
1101
Bits to Append: 001
Codeword = 1010001
ENTER THE MISMATCH OR CORRECT STRING:
1010001
Receiver Status = Data Properly Transmitted...
```

Program Code(Hamming Code):-

```
import java.util.*;

public class HammingCode
{

   public static int[] encode(int data[])
   {
      int code[] = new int[2 * data.length - 1];

      // Assign data bits
      code[2] = data[0];
      code[4] = data[1];
      code[5] = data[2];
      code[6] = data[3];

      // Calculate parity bits
      code[0] = code[2] ^ code[4] ^ code[6];
      code[1] = code[2] ^ code[5] ^ code[6];
      code[3] = code[4] ^ code[5] ^ code[6];

      return code;
   }

   public static void decode(int data[])
   {
      // Recalculate parity checks
      int p1 = data[0] ^ data[2] ^ data[4] ^ data[6];
      int p2 = data[1] ^ data[2] ^ data[5] ^ data[6];
      int p4 = data[3] ^ data[4] ^ data[5] ^ data[6];
```

```
        // Syndrome value = error position
        int syn = p4 * 4 + p2 * 2 + p1 * 1;

        if (syn == 0)
        {
            System.out.println("No Error Detected");
        }


        else
        {
            System.out.println("Wrong Word : " + Arrays.toString(data));
            System.out.println("Error Was Detected at pos: " + syn);

            // Correct the bit
            data[syn - 1] = (data[syn - 1] == 0) ? 1 : 0;

            System.out.println("Corrected Word : " + Arrays.toString(data));
        }
    }



    public static void main(String[] args)
    {
        int data[] = {1, 1, 0, 1};

        System.out.println("Data Word = " + Arrays.toString(data));

        int codeword[] = encode(data);
        System.out.println("Code Word = " + Arrays.toString(codeword));

        // Introduce error at position 5 (for testing)
        codeword[5] = (codeword[5] == 0) ? 1 : 0;

        decode(codeword);
    }
}
```

Output:-

```
Data Word = [1, 0, 1, 0]
Code Word = [1, 0, 1, 1, 0, 1, 0]
Wrong Word : [1, 0, 1, 1, 0, 0, 0]
Error Was Detected at pos: 6
Corrected Word : [1, 0, 1, 1, 0, 1, 0]
```