5. Write a program for Hamming Code generation for error detection and correction.

```
#include <stdio.h>

int main() {
   // Arrays to hold 7-bit encoded data and received data
   int data[7], recv[7];

   // Variables for parity bits (p1, p2, p3)
   // and parity check bits (c1, c2, c3)
   // 'c' will store the combined error position
   int p1, p2, p3, c1, c2, c3, c, i;

   //Input the 4 data bits from the user
   printf("Enter 4 bits of data: ");

   // We store them in positions 3, 5, 6, and 7
   // because positions 1, 2, and 4 are reserved for parity bits
   scanf("%d %d %d %d", &data[2], &data[4], &data[5], &data[6]);

   // Calculate the three parity bits using even parity
   // Parity bit 1 (position 1) covers bits 1, 3, 5, 7 → data[0], data[2], data[4], data[6]
   data[0] = data[2] ^ data[4] ^ data[6];

   // Parity bit 2 (position 2) covers bits 2, 3, 6, 7 → data[1], data[2], data[5], data[6]
   data[1] = data[2] ^ data[5] ^ data[6];

   // Parity bit 3 (position 4) covers bits 4, 5, 6, 7 → data[3], data[4], data[5], data[6]
   data[3] = data[4] ^ data[5] ^ data[6];

   // Display the final 7-bit Hamming code
   printf("\nEncoded data: ");
   for (i = 0; i < 7; i++)
      printf("%d", data[i]);
   printf("\n");

   // Simulate reception — enter the received 7-bit data
   printf("\nEnter received 7 bits:");
   for (i = 0; i < 7; i++)
      scanf("%d", &recv[i]);
```

```
    // Recalculate the parity bits from the received data to check for errors

    // Check bit 1 covers positions 1,3,5,7
    c1 = recv[0] ^ recv[2] ^ recv[4] ^ recv[6];

    // Check bit 2 covers positions 2,3,6,7
    c2 = recv[1] ^ recv[2] ^ recv[5] ^ recv[6];

    // Check bit 3 covers positions 4,5,6,7
    c3 = recv[3] ^ recv[4] ^ recv[5] ^ recv[6];

    // Combine check bits into a single binary value
    // This gives the position of the erroneous bit (if any)
    // c3 is the most significant bit, c1 is the least significant bit
    c = c3 * 4 + c2 * 2 + c1;

    // Display results
    if (c == 0) {
        // No parity errors → data received correctly
        printf("\nNo error detected in received data.\n");
    } else {
        // 'c' tells the bit position (1–7) that has the error
        printf("\nError detected at bit position: %d\n", c);

        // Correct the error by flipping that bit
        // If the bit is 0, make it 1; if it is 1, make it 0
        recv[c - 1] = (recv[c - 1] == 0) ? 1 : 0;

        // Display corrected code
        printf("\nCorrected data: ");
        for (i = 0; i < 7; i++)
            printf("%d", recv[i]);
        printf("\n");
    }

    return 0;
}
```

OUTPUT:

```
Enter 4 bits of data: 1 0 1 1

Encoded data: 0110011

Enter received 7 bits: 0110111

Error detected at bit position: 4

Corrected data: 0110011
```