Experiment No:2

Aim: WAP to implement Cyclic Redundancy Check & Hamming Code for Error Detection

Theory:

Error:A condition when the receiver's information does not matches with the sender's information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from sender to receiver. That means a 0 bit may change to 1 or a 1 bit may change to 0.
Error Detecting Codes (Implemented either at Data link layer or Transport Layer of OSI Model):
Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if any error has occurred during transmission of the message.
Basic approach used for error detection is the use of redundancy bits, where additional bits are added to facilitate detection of errors.
Some popular techniques for error detection are:
1. Simple Parity check
2. Two-dimensional Parity check
3. Checksum
4. Cyclic redundancy check
5. Hamming code

A]WAP to implement Cyclic Redundancy Check

The cyclic redundancy check (CRC) was invented by W. Wesley Peterson in 1961.

This technique is used to protect blocks of data called Frames. Using this technique, the transmitter appends an extra n- bit sequence to every frame called Frame Check Sequence (FCS). The FCS holds redundant information about the frame that helps the transmitter detect errors in the frame.

A CRC is commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption.

Features:

- CRCs are so called because the check (data verification) value is a redundancy (it adds zero information i.e. it expands the message without adding information) and the algorithm is based on cyclic codes.
- The CRC is a very powerful but easily implemented technique to obtain data reliability.
- It is a non-secure hash function designed to detect accidental changes to raw computer data.
- It is commonly used in digital networks and storage devices such as hard disk drives.

Advantages:

1. CRCs are simple to implement in binary hardware
2. Easy to analyze mathematically

3. They are good at detecting common errors caused by noise in transmission channels. Because the check value has a fixed length, the function that generates it is occasionally used as a hash function.
4. Little overhead

Disadvantages:

1. CRC is an easily reversible function, which makes it unsuitable for use in digital signatures.
2. CRCs are not, by themselves, suitable for protecting against intentional alteration of data. In authentication applications for data security: convenient mathematical properties make it easy to compute the CRC adjustment required to match any given change to the data.

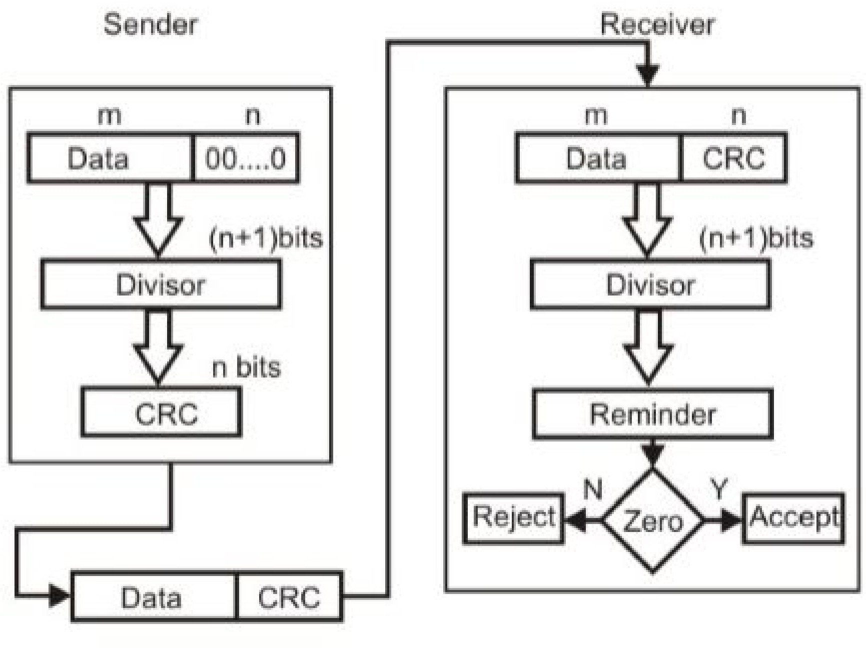
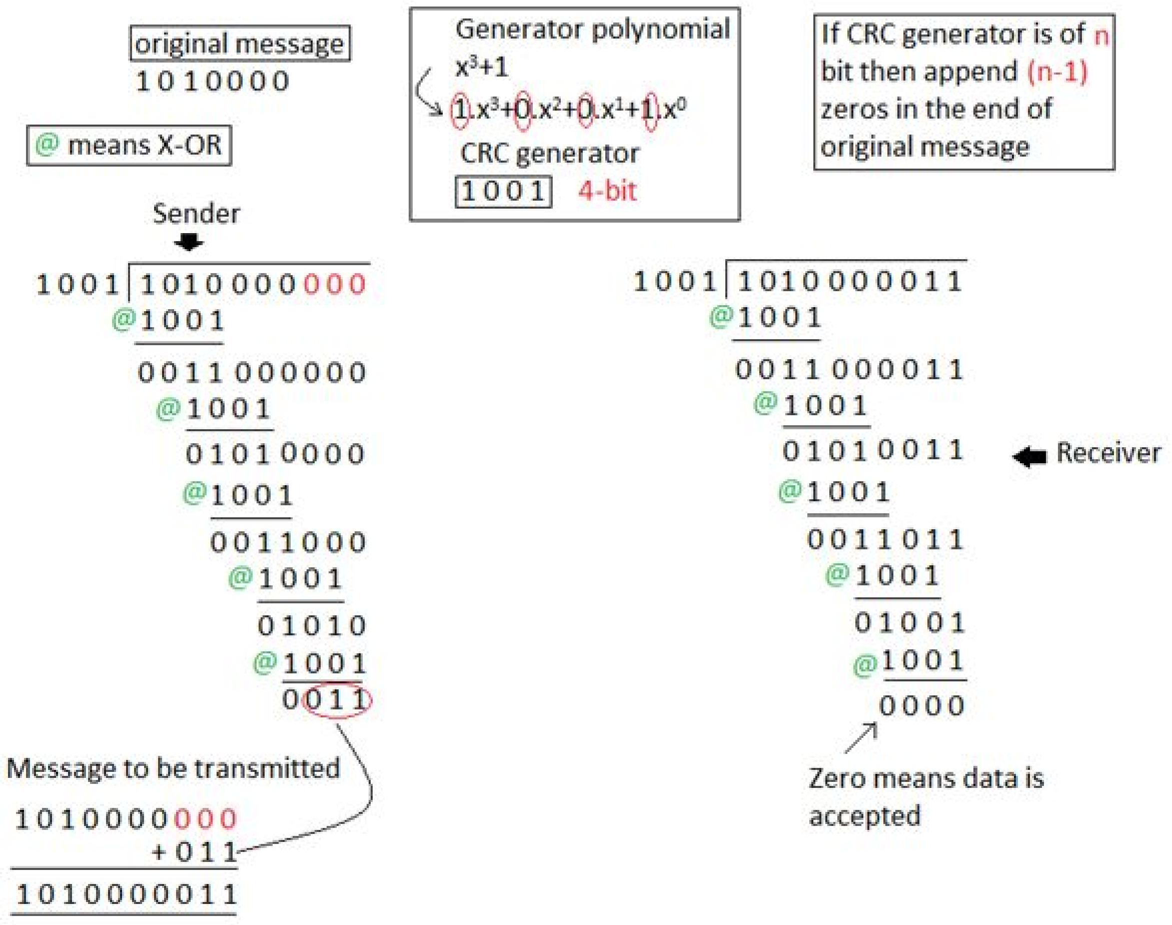


Figure 1: CRC

Example:

Assume the Original Data is 1010. Start with 4-bit generator polynomial/divisor: 1011. First append 3 additional bits (with value 000) on the end. Then calculate the crc (using exclusive-or):



The resulting crc is: 011 and the string that is sent is the original data appended with the crc: 1010000011. Suppose the following string of bits is received 1010000011 (Numbering from left to right, no bit has changed.) the receiver computes the crc.
The crc calculated is binary 0000, which means no error detected else error detected at particular bit position, which is the position of the bad bit.
steps:

- In checksum error detection scheme, the data is divided into k segments each of m bits.
- In the sender's end the segments are added using 1's complement arithmetic to get the sum. The sum is complemented to get the checksum.
- The checksum segment is sent along with the data segments.
- At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum. The sum is complemented.
- If the result is zero, the received data is accepted; otherwise discarded.

The most commonly-used checksum tools include:

- "cksum" - Unix commands generating 32-bit cyclic redundancy check (CRC) and byte count for an input file
- "md5sum" - Unix command generating Message-Digest Algorithm 5 (MD5) sum
- "jdigest" - Java GUI tool generating MD5 and Secure Hash Algorithm (SHA) sums
- "Jacksum" - Java application programming interface that incorporates numerous checksum implementations and permits any number of extensions
- "jcksum" - Java libraries used to calculate checksum using different algorithms

B) WAP to implement Hamming code for error detection and correction

Theory:

- It was invented by Richard Hamming in 1950
- Hamming code is a set of error-correction code that can be used to detect and correct bit errors that can occur when computer data is moved or stored.
- Hamming code can be applied to data units of any length and uses the relationship between data and redundancy bits.
- The number of parity bits required depends on the number of bits in the data transmission, and is calculated by the Hamming rule:

where, p is number of parity, m is length of the message.

- A 7 bit binary code requires 4 parity bits that can be added to the end of the data unit or interspersed with the original data bits. (parity = 3, message length = 4, binary code length = 3 + 4 = 7 bits)
- These bits are placed in positions 1,2,4 and 8. We refer to these bits as p1, p2, p4, p8.
- Position of parity bits in hamming code is shown below(right to left):

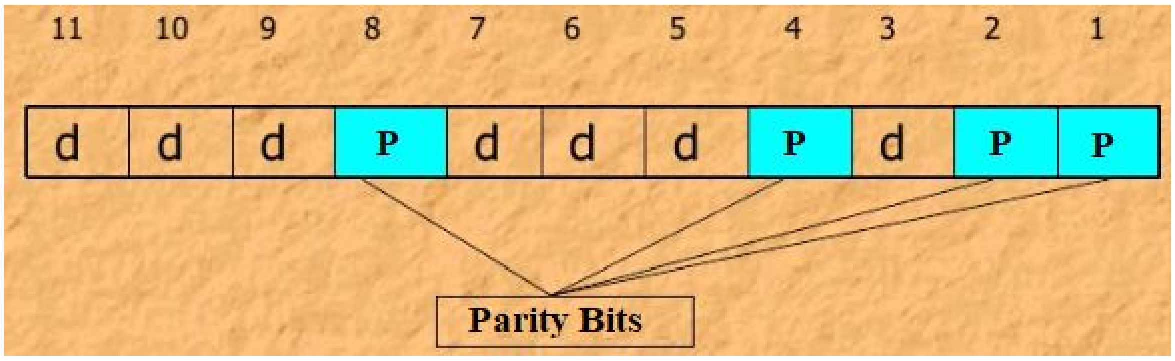


Figure 1: Parity and data bits

- In the hamming code, each p bit is combination of data bits:
  - p1 is one combination of data bits
  - p2 is two combination of data bits
  - p4 is four combination of data bits and so on.
- The combination used to calculate each of the 4 values for a 7 bit data sequence are as follows:
  - p1: bits 1,3,5,7,9,11
  - p2: bits 2,3,6,7,10,11
  - p4: bits 4,5,6,7
  - p8: bits 8,9,10,11
- If number of ones in finding parity (p = 1, 2, 4) is even then parity value is 0.
- Else if number of ones in finding parity (p = 1, 2, 4) is odd then parity value is 1.
  - To calculate even parity, the XOR operator is used; to calculate odd parity, the XNOR operator is used.
  - Hamming codes detect two bit errors by using more than one parity bit, each of which is computed on different combinations of bits in the data.

- The validity of the data can be checked when it is read or after it has been received in a data transmission. Using more than one parity bit, an error-correction code can not only identify a single bit error in the data unit, but also its location in the data unit.

Program:

A].CRC

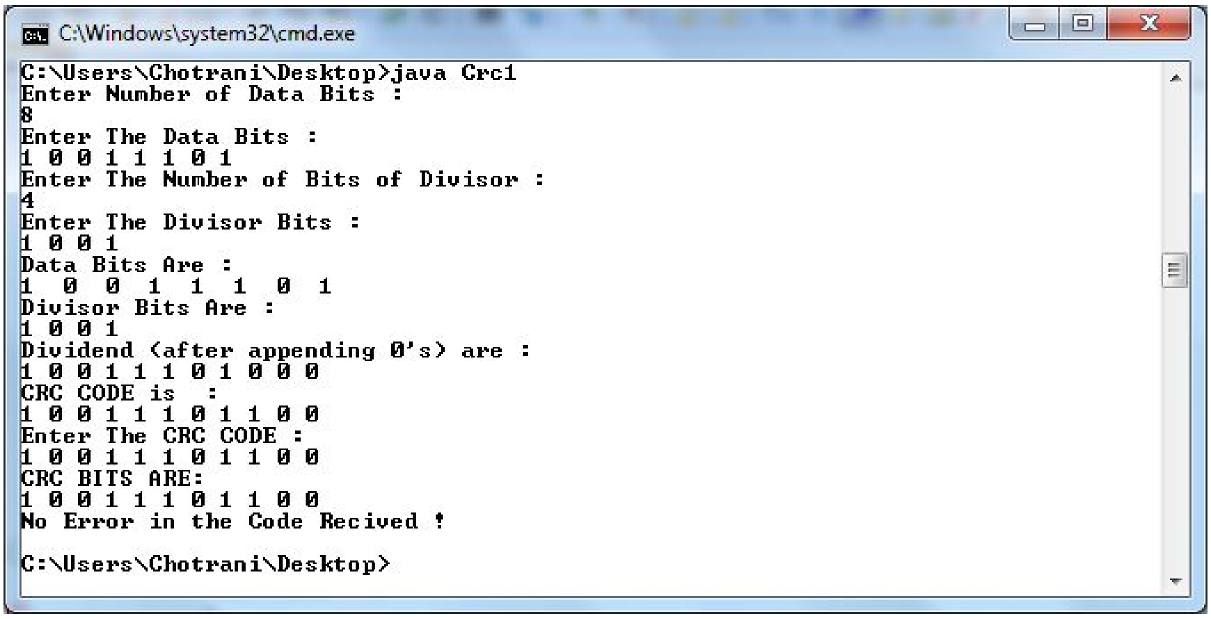
```
import java.util.*;
class Crc1
{
  public static void main(String[] args)
  {
    int[] data;
    int[] div;
    int[] divisor;
    int[] rem;
    int[] crc;
    int data_bits,divisor_bits,tot_bits;
    Scanner s = new Scanner(System.in);
    //Sender Side
    System.out.println("Enter Number of Data Bits :");
    data_bits= s.nextInt();
    data= new int[data_bits];
    System.out.println("Enter The Data Bits : ");
    for(int i=0;i<data_bits;i++)
      data[i] = s.nextInt();
System.out.println("Enter The Number of Bits of Divisor");
    divisor_bits= s.nextInt();
    divisor = new int[divisor_bits];
    System.out.println("Enter The Divisor Bits : ");
    for(int i=0; i<divisor_bits ; i++)
      divisor[i] = s.nextInt();
    System.out.println("Data Bits Are :");
    for(int i=0; i<data_bits ; i++)
      System.out.print(data[i] + "  ");
    System.out.println("\nDivisor Bits Are :");
    for(int i=0; i<divisor_bits; i++)
      System.out.print(divisor[i] + " ");
    tot_bits = data_bits+divisor_bits-1;
    div = new int[tot_bits];
    rem = new int[tot_bits];
    crc = new int[tot_bits];
  for(int i=0;i<data.length;i++)
    div[i] = data[i];
  System.out.println("\nDividend (after appending 0's) are”);
  for(int i=0;i<div.length;i++)
    System.out.print(div[i]+" ");
  System.out.print("\n");
  for(int j=0;j<div.length ;j++)
    rem[j] = div[j];
  rem = divide(div,divisor,rem);
  for(int i=0;i<div.length;i++)
    crc[i] = (div[i] ^ rem[i]);
  System.out.println("CRC CODE is  :");
  for(int i=0;i<crc.length;i++)
    System.out.print(crc[i] + " ");

  //Reciver Side
  System.out.println("\nEnter The CRC CODE :");
  for(int i=0;i<crc.length;i++)
  crc[i] = s.nextInt();
  System.out.println("CRC BITS ARE:");
  for(int i =0; i<crc.length ; i++)
    System.out.print(crc[i] + " ");
  for(int j=0; j<crc.length; j++)
    rem[j] = crc[j];
  rem = divide(crc,divisor,rem);
  for(int i=0;i<rem.length; i++)
  {
    if(rem[i]!=0)
    {
       System.out.println("\nError in the Code recievd!");
      break;
    }
    if(i==rem.length-1)
          System.out.print("\nNo Error in the Code

  }
}
static int[] divide(int div[], int divisor[], int rem[])
{
  int cur=0;
```

```
    while(true)
    {
      for(int i=0; i<divisor.length;i++)
        rem[cur+i] = (rem[cur+i] ^ divisor[i] );
      while(rem[cur] == 0 && cur!=rem.length-1)
        cur++;
      if((rem.length-cur)<divisor.length)
        break;
    }
    return rem;
  }
}
```

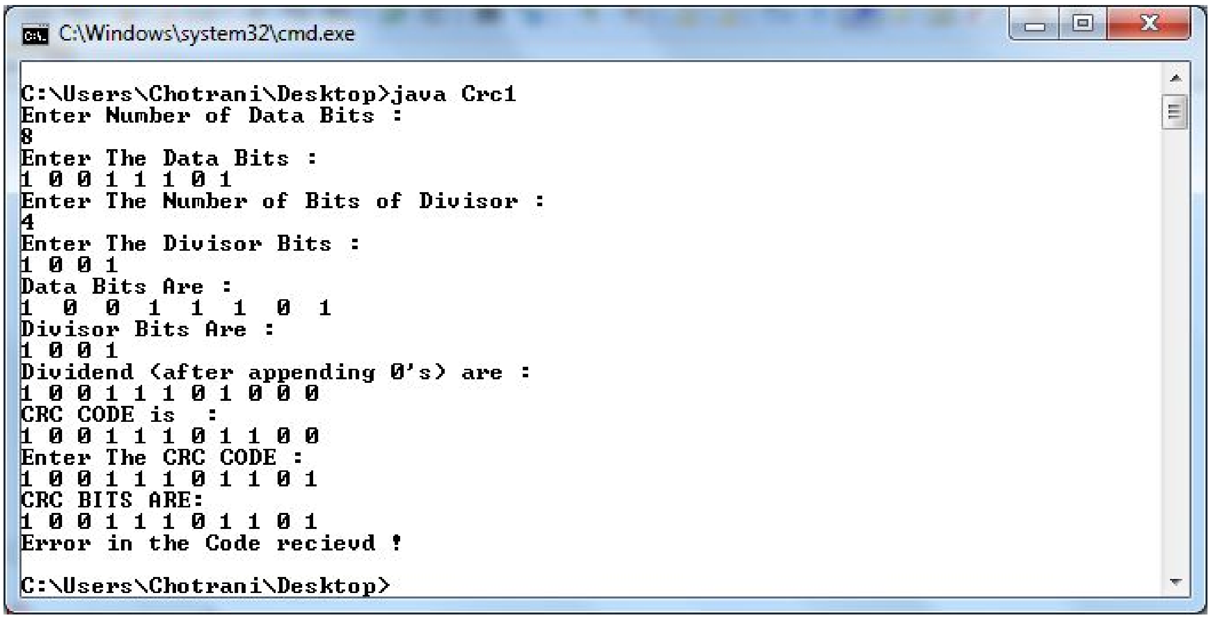
OUTPUT:



C:\Windows\system32\cmd.exe

```
C:\Users\Chotrani\Desktop>java Crc1
Enter Number of Data Bits :
8
Enter The Data Bits :
1 0 0 1 1 1 0 1
Enter The Number of Bits of Divisor :
4
Enter The Divisor Bits :
1 0 0 1
Data Bits Are :
1  0  0  1  1  1  0  1
Divisor Bits Are :
1 0 0 1
Dividend (after appending 0's) are :
1 0 0 1 1 1 0 1 0 0 0
CRC CODE is  :
1 0 0 1 1 1 0 1 1 0 0
Enter The CRC CODE :
1 0 0 1 1 1 0 1 1 0 0
CRC BITS ARE:
1 0 0 1 1 1 0 1 1 0 0
No Error in the Code Recived !

C:\Users\Chotrani\Desktop>
```



C:\Windows\system32\cmd.exe

```

C:\Users\Chotrani\Desktop>java Crc1
Enter Number of Data Bits :
8
Enter The Data Bits :
1 0 0 1 1 1 0 1
Enter The Number of Bits of Divisor :
4
Enter The Divisor Bits :
1 0 0 1
Data Bits Are :
1  0  0  1  1  1  0  1
Divisor Bits Are :
1 0 0 1
Dividend (after appending 0's) are :
1 0 0 1 1 1 0 1 0 0 0
CRC CODE is  :
1 0 0 1 1 1 0 1 1 0 0
Enter The CRC CODE :
1 0 0 1 1 1 0 1 1 0 1
CRC BITS ARE:
1 0 0 1 1 1 0 1 1 0 1
Error in the Code recievd !

C:\Users\Chotrani\Desktop>
```

B]Hamming

```
import java.util.*;
class hamming
{
  public static void main(String args[])throws Exception
  {
    Scanner read=new Scanner(System.in);
    int i,value;
    int a[] = new int[7];
    int r[] = new int[7];
    int v[] = new int[3];
    a[0] = -1;
    a[1] = -1;
    a[3] = -1;
    value = 0;
    System.out.println("Enter data in bits:");
    for(i=6;i>=0;i--)
      if(i!=3 && i!=1 && i!=0)
        a[i]=read.nextInt();
    int c;

    //calculating parity 1
    c=0;
    if(a[2]==1)
      c++;
    if(a[4]==1)
      c++;
    if(a[6]==1)
      c++;
    if(c%2==0)
      a[0]=0;
    else
      a[0]=1;

    //calculating parity 2
    c=0;
    if(a[2]==1)
      c++;
    if(a[5]==1)
      c++;
```

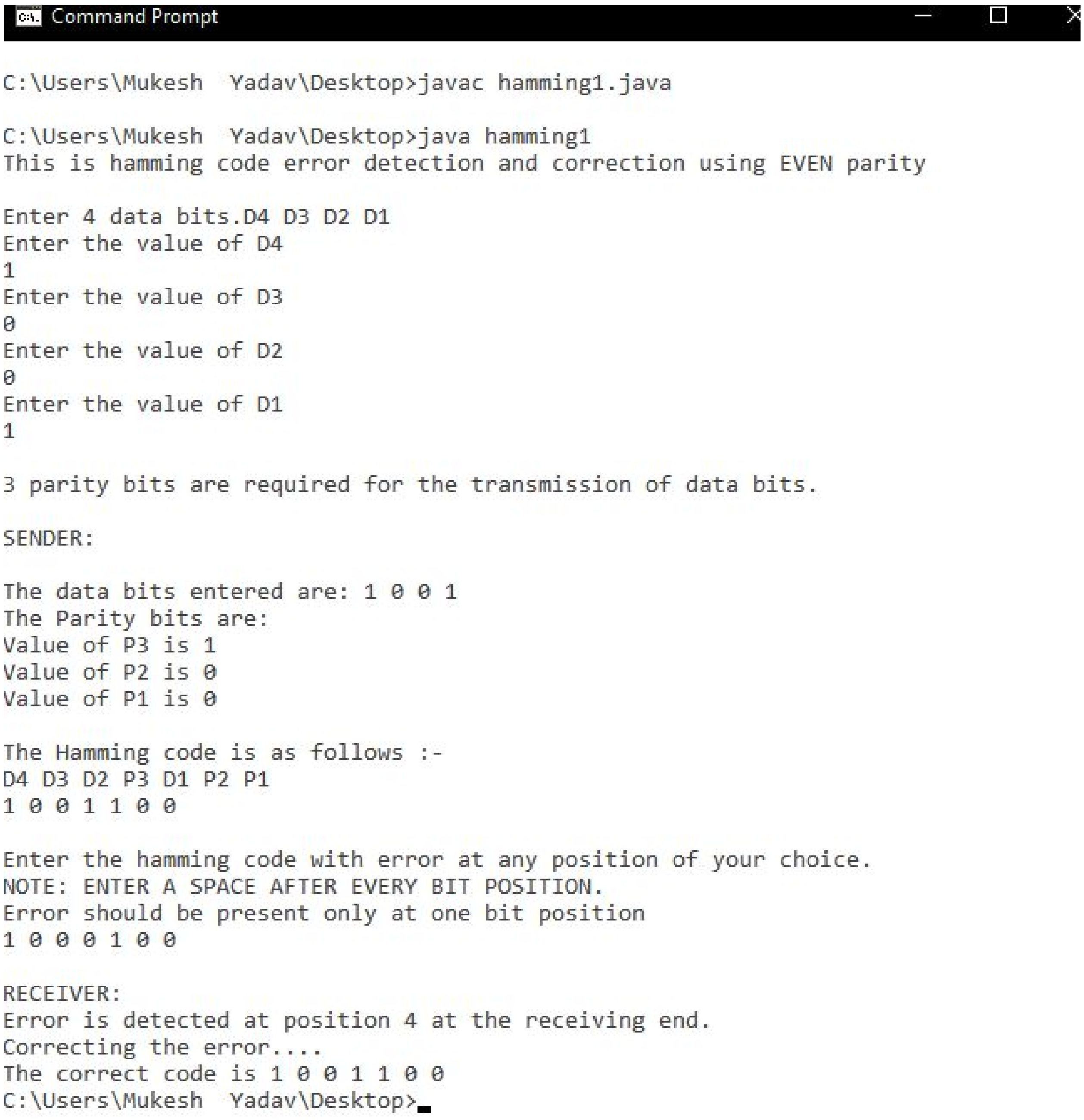
```
if(a[6]==1)
  c++;
if(c%2==0)
  a[1]=0;
else
  a[1]=1;
//calculating parity 3
c=0;
if(a[4]==1)
  c++;
if(a[5]==1)
  c++;
if(a[6]==1)
  c++;
if(c%2==0)
  a[3]=0;
else
  a[3]=1;

       System.out.println("The encoded message is:");
for(i=6;i>=0;i--)
  System.out.println(a[i]);
System.out.println();
System.out.println("Received message:");
for(i=6;i>=0;i--)
  r[i]=read.nextInt();
c=0;
if(r[0]==1)
  c++;
if(r[2]==1)
  c++;
if(r[4]==1)
  c++;
if(r[6]==1)
  c++;
if(c%2==0)
  v[0]=0;
else
  v[0]=1;

c=0;
if(r[1]==1)
```

```
          c++;
        if(r[2]==1)
          c++;
        if(r[5]==1)
          c++;
        if(r[6]==1)
          c++;
        if(c%2==0)
          v[1]=0;
        else
          v[1]=1;
          c=0;
        if(r[3]==1)
          c++;
        if(r[4]==1)
          c++;
        if(r[5]==1)
          c++;
        if(r[6]==1)
          c++;
        if(c%2==0)
          v[2]=0;
        else
          v[2]=1;
        value=v[0]+v[1]*2+v[2]*4;
        if(value==0)
          System.out.println("NO ERROR!!");
        else
        {
System.out.println("Error detected at: " +value+ "th position");
          if(r[value-1]==0)
            r[value-1]=1;
          else
            r[value-1]=0;
          System.out.println("Corrected message:");
          for(i=6;i>=0;i--)
            System.out.println(r[i]);
        }
      }
    }
```

Output:



```
C:\Users\Mukesh  Yadav\Desktop>javac hamming1.java

C:\Users\Mukesh  Yadav\Desktop>java hamming1
This is hamming code error detection and correction using EVEN parity

Enter 4 data bits.D4 D3 D2 D1
Enter the value of D4
1
Enter the value of D3
0
Enter the value of D2
0
Enter the value of D1
1

3 parity bits are required for the transmission of data bits.

SENDER:

The data bits entered are: 1 0 0 1
The Parity bits are:
Value of P3 is 1
Value of P2 is 0
Value of P1 is 0

The Hamming code is as follows :-
D4 D3 D2 P3 D1 P2 P1
1 0 0 1 1 0 0

Enter the hamming code with error at any position of your choice.
NOTE: ENTER A SPACE AFTER EVERY BIT POSITION.
Error should be present only at one bit position
1 0 0 0 1 0 0

RECEIVER:
Error is detected at position 4 at the receiving end.
Correcting the error....
The correct code is 1 0 0 1 1 0 0
C:\Users\Mukesh  Yadav\Desktop>
```

Conclusion: Hence we successfully studied the program of CRC and hamming of error detection and correction.

Date:

Sign: **Grade:**