

### Program 3

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

Java is a general-purpose computer programming language that is simple, concurrent, class-based, object-oriented language. The compiled Java code can run on all platforms that support Java without the need for recompilation hence Java is called as "write once, run anywhere" (WORA). The Java compiled intermediate output called "byte-code" that can run on any Java virtual machine (JVM) regardless of computer architecture. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

In Linux operating system Java libraries are preinstalled. It's very easy and convenient to compile and run Java programs in Linux environment. To compile and run Java Program is a two-step process:

1. Compile Java Program from Command Prompt

**[root@host ~]# javac Filename.java**

The Java compiler (Javac) compiles java program and generates a byte-code with the same file name and .class extension.

2. Run Java program from Command Prompt

**[root@host ~]# java Filename** The java interpreter (Java) runs the byte-code and gives the respective output. It is important to note that in above command we have omitted the .class suffix of the byte- code (Filename.class).

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check. To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The CRC calculation or cyclic redundancy check was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC. Also each data block on your hard disk has a CRC value attached to it. Modern computer world cannot do without these CRC calculations. So let's see why they are so widely used. The answer is simple; they are powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips are used. The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school. We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation—by dividing it by 19 or 10011. Please note that 19 is an odd number. This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you were young. It might only look a little bit strange. Also notations differ between countries, but the method is similar. With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "Lammert"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient. All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be

equal to an exclusive or operation. Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always the same:

- The message bits are appended with  $c$  zero bits; this augmented message is the dividend
- A predetermined  $c+1$ -bit binary sequence, called the generator polynomial, is the divisor
- The checksum is the  $c$ -bit remainder that results from the division operation

Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs.

Remember that the width of the divisor is always one bit wider than the remainder. So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.

```
import java.io.*;
class Crc
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int[ ] data;
int[ ]div;
int[ ]divisor;
int[ ]rem;
int[ ] crc;
int data_bits, divisor_bits, tot_length;
System.out.println("Enter number of data bits : ");
data_bits=Integer.parseInt(br.readLine());
data=new int[data_bits];
System.out.println("Enter data bits : ");
for(int i=0; i<data_bits; i++)
data[i]=Integer.parseInt(br.readLine());
System.out.println("Enter number of bits in divisor : ");
divisor_bits=Integer.parseInt(br.readLine());
divisor=new int[divisor_bits];
System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
divisor[i]=Integer.parseInt(br.readLine());
/* System.out.print("Data bits are : ");
for(int i=0; i< data_bits; i++)
System.out.print(data[i]);
System.out.println();
System.out.print("divisor bits are : ");
for(int i=0; i< divisor_bits; i++)
System.out.print(divisor[i]);
System.out.println();
*/ tot_length=data_bits+divisor_bits-1;
div=new int[tot_length];
rem=new int[tot_length];
crc=new int[tot_length];
/*----- CRC GENERATION-----*/
for(int i=0;i<data.length;i++)
div[i]=data[i];
System.out.print("Dividend (after appending 0's) are : ");
```

```

for(int i=0; i< div.length; i++)
System.out.print(div[i]);
System.out.println();
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++) //append dividend and remainder
{
crc[i]=(div[i]^rem[i]);
}
System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
System.out.print(crc[i]);
/*-----ERROR DETECTION-----*/
System.out.println();
System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(int i=0; i<crc.length; i++)
crc[i]=Integer.parseInt(br.readLine());
/* System.out.print("crc bits are : ");
for(int i=0; i< crc.length; i++)
System.out.print(crc[i]);
System.out.println();
*/
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("Error");
break;
}
if(i==rem.length-1)
System.out.println("No Error");
}
System.out.println("THANK YOU.... :)");
}
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)

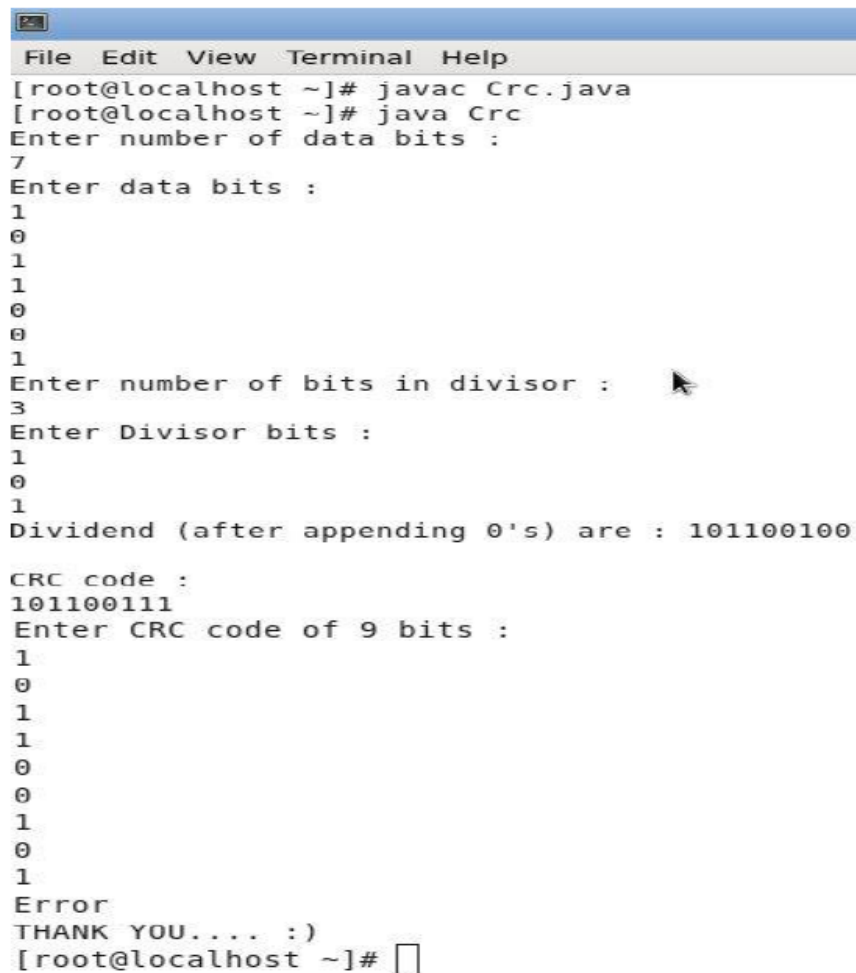
```

```

cur++;
if((rem.length-cur)<divisor.length)
break;
}
return rem;
}
}

```

Output:



```

File Edit View Terminal Help
[root@localhost ~]# javac Crc.java
[root@localhost ~]# java Crc
Enter number of data bits :
7
Enter data bits :
1
0
1
1
0
0
1
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Dividend (after appending 0's) are : 101100100

CRC code :
101100111
Enter CRC code of 9 bits :
1
0
1
1
0
0
1
0
1
Error
THANK YOU.... :)
[root@localhost ~]#

```

## Code Explanation

```
import java.io.*;  
class Crc {
```

This line imports the necessary Java input/output classes and defines a class named `Crc`.

```
public static void main(String args[]) throws IOException {
```

This line declares the main method. The `throws IOException` indicates that the method may throw an IOException, so it needs to be handled or declared.

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Here, a BufferedReader object `br` is created to read input from the console.

```
int[] data;  
int[] div;  
int[] divisor;  
int[] rem;  
int[] crc;
```

These lines declare arrays to store data, divisor, remainder, and CRC values.

```
int data_bits, divisor_bits, tot_length;
```

Variables `data\_bits`, `divisor\_bits`, and `tot\_length` are declared to store the number of data bits, number of bits in the divisor, and the total length.

```
System.out.println("Enter number of data bits : ");  
data_bits = Integer.parseInt(br.readLine());
```

The program prompts the user to enter the number of data bits and reads the input from the console.

```
data = new int[data_bits];  
System.out.println("Enter data bits : ");  
for (int i = 0; i < data_bits; i++)  
    data[i] = Integer.parseInt(br.readLine());
```

An array `data` is created to store the data bits, and the program prompts the user to enter the data bits, which are then read and stored in the array.

```
System.out.println("Enter number of bits in divisor : ");  
divisor_bits = Integer.parseInt(br.readLine());
```

The program prompts the user to enter the number of bits in the divisor and reads the input.

```
divisor = new int[divisor_bits];  
System.out.println("Enter Divisor bits : ");  
for (int i = 0; i < divisor_bits; i++)  
    divisor[i] = Integer.parseInt(br.readLine());
```

An array `divisor` is created to store the divisor bits, and the program prompts the user to enter the divisor bits, which are then read and stored in the array.

```

tot_length = data_bits + divisor_bits - 1;
div = new int[tot_length];
rem = new int[tot_length];
crc = new int[tot_length];

```

The total length is calculated, and arrays `div`, `rem`, and `crc` are initialized with the calculated length.

```

/*----- CRC GENERATION-----*/
for (int i = 0; i < data.length; i++)
    div[i] = data[i];

```

The data bits are copied to the `div` array.

```

System.out.print("Dividend (after appending 0's) are : ");
for (int i = 0; i < div.length; i++)
    System.out.print(div[i]);
System.out.println();

```

The program prints the dividend (data bits after appending 0's) to the console.

```

for (int j = 0; j < div.length; j++) {
    rem[j] = div[j];
}

```

The remainder array is initialized with the values of the dividend.

```

rem = divide(div, divisor, rem);

```

The `divide` method is called to perform the division.

```

for (int i = 0; i < div.length; i++) {
    crc[i] = (div[i] ^ rem[i]);
}

```

The CRC code is generated by XORing the original data bits with the remainder.

```

System.out.println();
System.out.println("CRC code : ");
for (int i = 0; i < crc.length; i++)
    System.out.print(crc[i]);

```

The generated CRC code is printed to the console.

```

/*-----ERROR DETECTION-----*/
System.out.println();
System.out.println("Enter CRC code of " + tot_length + " bits : ");
for (int i = 0; i < crc.length; i++)
    crc[i] = Integer.parseInt(br.readLine());

```

The program prompts the user to enter the CRC code for error detection.

```

for (int j = 0; j < crc.length; j++) {

```

```

    rem[j] = crc[j];
}
rem = divide(crc, divisor, rem);

```

The remainder array is initialized with the entered CRC code, and the `divide` method is called to perform the division.

```

for (int i = 0; i < rem.length; i++) {
    if (rem[i] != 0) {
        System.out.println("Error");
        break;
    }
    if (i == rem.length - 1)
        System.out.println("No Error");
}
System.out.println("THANK YOU.... :)");

```

The program checks for errors in the remainder and prints whether an error is detected or not. Finally, a thank you message is printed.

```

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;
}

```

The `divide` method performs the polynomial division and returns the remainder. The XOR operation is used to update the remainder during the division process. The method is called with the dividend, divisor, and initial remainder arrays.