Shree Rahul Education Society's (Regd.)
SHREE L. R. TIWARI COLLEGE OF ENGINEERING
Kanakia Park, Near Commissioner's Bungalow, Mira Road (East), Thane 401107, Maharashtra
(Approved by AICTE, Govt. of Maharashtra & Affiliated to University of Mumbai)
NAAC Accredited | ISO 9001:2015 Certified
Tel. No.: 022-28120144 / 022-28120145 | Email: slrtce@rahuleducation.com | Website: www.slrtce.in

DEPARTMENT OF COMPUTER ENGINEERING

CSL502Computer Network Laboratory

Fifth Semester, 2020-2021 (Odd Semester)

Name of Student : Neha Mahendra Yadav

Roll No. : 67

Division :TE-CMPN

Batch :B3

Day / Session : Tuesday/Afternoon

Venue : google meet

Experiment No. 06

Title of Experiment : To implement Hamming Code and CRC using Java.

Date of Conduction :

Date of Submission :

Particulars	Max. Marks	Marks Obtained
Preparedness and Efforts(PE)	3	
Knowledge of tools(KT)	3	
Debugging and results(DR)	3	
Documentation(DN)	3	
Punctuality & Lab Ethics(PL)	3	
Total	15	

Grades – Meet Expectations (3 Marks), Moderate Expectations (2 Marks), Below Expectations (1 Mark

Checked and Verified by

Name of Faculty : Prof. Rajesh Gaikwad

Signature :

Date : 4/08/2020

EXPERIMENT 6

Title: To implement Hamming Code and CRC using Java.

Objectives:

1) Hamming Code:

1) A concise method of detecting pre location of an error so it can be detected and corrected without any drastic action. To detect single bit errors .

2) To correct single bit data which was detected as error bit . To analyze process and detect whether any error is present which can effect program execution. Safe and proper execution , since it not only detects but also solve the error for error free execution.

2) Cyclic Redundancy Check (CRC):

1) It is an error detection code like Hamming Code. To detect error using binary division method .It detects error at bits also known as Cyclic Redundancy Check Bits. The Redundancy Bits are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.

2)It is an error implementation code used for proper execution of any given process without errors. It also allows us to use new technique for detecting error using binary division method.

Pre Concepts:

1) Hamming Code: Hamming code is a set of error-correction code s that can be used to detect and correct bit errors that can occur when computer data is moved or stored. Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer.

Parity bits –
A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1’s in the data are even or odd. Parity bits are used for error detection. There are two types of parity bits:

Even parity bit:

In the case of even parity, for a given set of bits, the number of 1's are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1's an even number. If the total number of 1's in a given set of bits is already even, the parity bit's value is 0.

Odd Parity bit –

In the case of odd parity, for a given set of bits, the number of 1's are counted. If that count is even, the parity bit value is set to 1, making the total count of occurrences of 1's an odd number. If the total number of 1's in a given set of bits is already odd, the parity bit's value is 0.

3) Java Implemention: The above both programs are executed in java very easily and efficiently. The above process can be executed in both linux or normal Window Operating System. Basic Java compilation and concepts are required for implementing both above processes.

New Concepts:

1) Hamming Code: Easy execution of hamming code with the help of Java Programing, without any extra work. After executing Hamming code through java , it was analyzed that hamming code is very essential error detecting code. In hamming code every single bits is detected for error, and it also becomes very easy to detect and solve that particular error bit. It also provides us knowledge that how hamming code algorithim is implemented in Computer Netwoks. By java programing it also makes us understand the algorithim perfectly through java code.

2) Cyclic Redundancy Check(CRC): We will learn how to detect Control Error, Detection and Correction in Data link layer in Computer Networking. It also makes us learn the CHKSUM error and how to implement this kind of errors. Provide knowledge of polynomial division of 16 or 32 bits in length. CRC method also provides an accuracy execution. They are very applicable for detecting errors during transmission. It also provides little protection to the data. Implemention of these process using java programing gives an virtual visualization that how this particular algorithim works and detect as well as solve errors.

The both above algorithim was introduced in order to detect and solve errors. Both have same operation but works with different applications. By Java Programing the both the techniques can be easily understood and also can be executed. The execution of both the above process gives us an rough idea that how error are detected differently and errors are solved. Both the above techniques is very essential and effective in their respective work.

Program Code:

1) The program for implementation of Hamming Code using Java is given

below: import java.io.*;

```
class HammingCode
{
 public static void main(String args[])throws IOException
 {
 BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("<<====>>");

System.out.println(" Program Created Br SuRjEEt");

System.out.println("<<====>>");
System.out.println("Enter No of Data To be Transmitted");
int d=Integer.parseInt(br.readLine());
 parity pa=new parity();
 int i,j,sum,k;

System.out.println("<<====>>");
int count=0;
System.out.println("Enter Parity");
System.out.println("1.Odd");
System.out.println("2.Even");
 int parity1=Integer.parseInt(br.readLine());


 int p=pa.paritybit(d);

  int array[]=new int[9999];

 int hamming[]=new int[p+d+1];

 for(i=1;i<hamming.length;i++)
 hamming[i]=0;

System.out.println("<<====>>");

System.out.println("Enter Data Elements As Per Increasing Order");


 int a[]=new int[d];

 for(i=0;i<d;i++)
 a[i]=Integer.parseInt(br.readLine());

 j=0;
```

```
 for(i=1;i<hamming.length;i++) {
 if(!pa.power2(i,hamming))
 hamming[i]=a[j++];


 }



 for(i=1;i<hamming.length;i++)
 { k=i;
 count=0;
 sum=0;


 if(pa.power2(i,hamming))
 count=pa.parcal(i, array, count,hamming);



j=0;
 while(j<count)
 {
 sum=sum+hamming[array[j]];
j++;

 }
 if(pa.power2(k,hamming))
 if(parity1==2)
 if(sum%2==0)
 hamming[k]=0;
 else
 hamming[k]=1;
 else
 if(sum%2==0)
 hamming[k]=1;
 else
 hamming[k]=0;
 }
 System.out.println("<<====>>");
 System.out.println(" Hamming Code Is As Shown:");
 System.out.print(" ");

 for(i=hamming.length-1;i>0;i--)
```

```
 System.out.print(hamming[i]+" ");
System.out.println();
System.out.println("<<====>>");
System.out.println(" Place An Error");
System.out.println("Enter position to be
inverted"); System.out.println("If Not Press 0");
int n=Integer.parseInt(br.readLine());
if(n<hamming.length)
if(hamming[n]==1)
 {
hamming[n]=0;
 }
 else
 {
hamming[n]=1;
 }
 else
 {
 System.out.println("Invalid Position. ...");

 System.out.println("Plzz Check Hamming Code");

 }
int newsum,m=0;
int error[]=new int[array.length];

for(i=1;i<hamming.length;i++)
 { k=i;
 count=0;
 sum=0;


 if(pa.power2(i,hamming))
 count=pa.parcal(i, array, count,hamming);

j=0;
newsum=hamming[k];

 while(j<count)
 {
 newsum=newsum+hamming[array[j]];
j++;

 }
 if(pa.power2(k,hamming))
 if(parity1==2)
 if(newsum%2==0)
```

```
error[m++]=0;
else
error[m++]=1;
else
if(newsum%2==0)
error[m++]=1;
else
error[m++]=0;
}

System.out.print(" Hamming Code
Sending"); for(int b=1;b<7;b++)
System.out.print(" . ");
System.out.println();
System.out.println();
System.out.println();

System.out.println();
System.out.println();
System.out.println();



System.out.print(" Hamming Code
Received!!!!!!"); System.out.println();


for(i=hamming.length-1;i>0;i--)
System.out.print(hamming[i]+" ");
System.out.println();



System.out.println();
System.out.println();
System.out.println();



int dec=0;
for(i=0;i<hamming.length;i++)
dec=dec+((int)(Math.pow(2,i)))*error[i];

if(dec==0)
System.out.println("No Error Detected In Hamming
Code"); else
System.out.println("Error Detected at location "+dec);
```

```
if(dec!=0)
 {
if(hamming[dec]==0)
hamming[dec]=1;
else
hamming[dec]=0;


System.out.println();
System.out.println();
System.out.println();



System.out.println("Corrected Code is");
System.out.print(" ");

for(i=hamming.length-1;i>0;i--)
System.out.print(hamming[i]+" ");
System.out.println();

 }

System.out.println("<<====>>");
 }

 }
class parity
{
 public int paritybit(int x)
 {

 int y=0,z=1,p=0;
 while(y<z)
 {
 y=(int)Math.pow(2,p);
 z=x+p+1;
 if(y<z)
 p++;

 }
 return p;

 }

 public boolean power2(int j,int h[])
```

```
{
int k;

for(k=0;k<h.length;k++)
{ if(j==(int)(Math.pow(2, k)))
return true;
}


return false;

}

public int parcal(int x,int array[],int count,int h[])
{

String str=binary(x);
int i=0;
int pos=0;
int n=0;

while(i<str.length())
{
if(str.charAt(i)=='1')
pos=i;
i++;

}

for(int k=x+1;k<h.length;k++)
{
str=binary(k);
if(str.charAt(pos)=='1')
{
array[n++]=k;
count++;
}
}

return count;

}


public String binary(int i)
{
String str="";
```

```
 String str1="";
 int count=0;
 String s;
 int c;

 while(i!=0)
 {
 int z=i%2;

 s=Integer.toString(z);

 str=str.concat(s); i=i/2;
 count++;
 }

 return str;
 }}
```

2) The program for implementation of Cyclic Redundancy Check(CRC) using Java is given below:

```
import java.util.*;
class CRC{
 public static void main(String sap[])
 {
 Scanner sc = new Scanner(System.in);
 int i,j,flag=0;

 //string input of dividend...
 System.out.print("\nEnter dividend: ");
 String dividend = sc.next();
 //string input for divisor...
 System.out.print("\nEnter divisor: ");
 String divisor = sc.next();
 //length of dividend and divisor...
 int dividend_length = dividend.length();
 int divisor_length = divisor.length();
 //array for dividend, divisor and send _array...
 int dividend_array[] = new int[dividend_length + divisor_length - 1];
int send_arr[] = new int[dividend_length + divisor_length - 1];
 int divisor_array[] = new int[divisor_length];

 //loading dividend values into dividend array...
 for(i=0;i<dividend_length;i++)
 {
 dividend_array[i] = Integer.parseInt(""+dividend.charAt(i));
 send_arr[i] = dividend_array[i];
 }
```

```
 for(i=dividend_length;i<dividend_array.length;i++)
 {
 dividend_array[i] = 0;
 }

 //loading divisor values into divisor array...
 for(i=0;i<divisor_length;i++)
 {
 divisor_array[i] = Integer.parseInt(""+divisor.charAt(i));
 }

 //temp array would hold temp dividend...
 int temp[] = new int[divisor_length];
 for(i=0;i<dividend_array.length;i++)
 {
 /*it would work only for '1' and for '0' it would do nothing just increment the index...*/
 if(dividend_array[i] == 1)
 {
 //loading temp dividend to temp array...
for(j=0;j<divisor_length;j++)
 {
 if(i+j>dividend_array.length-1) {
 //flag = 1 dividend < divisor flag=1;
 break;
 }
 else
 {
 temp[j] = dividend_array[i+j]; }
 }

 //flag = 0 means divison is possible...
if(flag==0)
 {
 for(j=0;j<divisor_length;j++)
 {
 if(temp[j] == divisor_array[j]) {
 dividend_array[i+j] = 0;
 }
 else
 {
 dividend_array[i+j] = 1;
 }
 }
 }
 }
 }
 for(i=dividend_length;i<dividend_array.length;i++) {
```

```
 send_arr[i] = dividend_array[i];
 }

 //displaying message to be transmitted...
System.out.print("\nMessage to be transmitted is: ");
for(i=0;i<send_arr.length;i++)
 {
 System.out.print(send_arr[i]);
 }
 } }
```

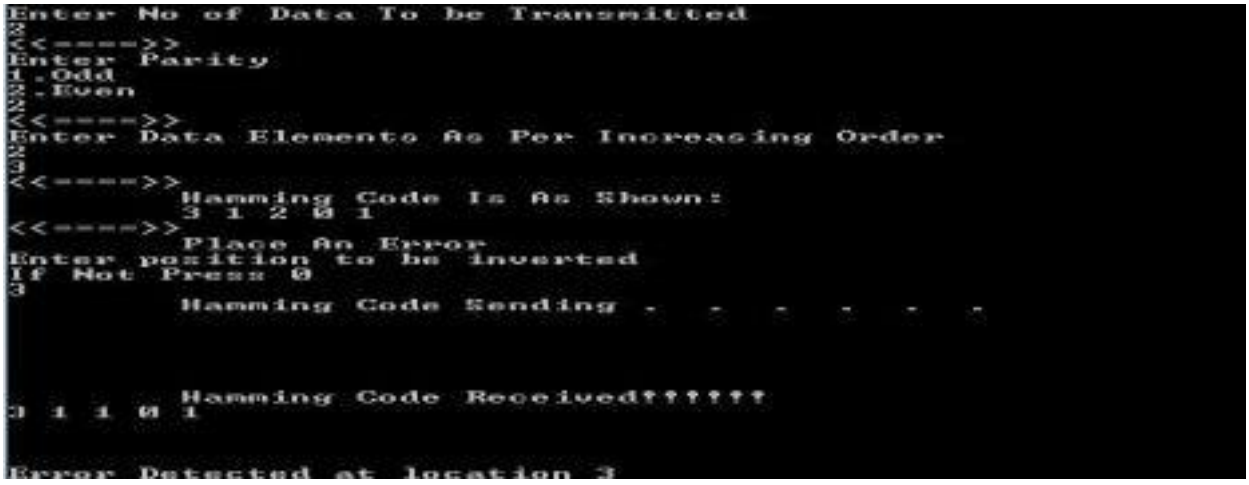
Outputs:

a) Hamming Code:

Here we first compile our java code for hamming code program:

After Compilation of our program we run the code: We have first Implemented For Odd Parity:

Now We Implement for Even Parity



b) Cyclic Redundancy Check (CRC):

Here we implement CRC code using java programing:

After compilation of the code , We execute CRC using java:

Here we provide the requires input , in order to obtain appropriate output:

Then we provide dividend less than the divisor:

Output:

1)Hamming code:

a)odd parity:

```
<<====>>
Enter No of Data To be Transmitted
4
<<====>>
Enter Parity
1.Odd
2.Even
1
<<====>>
Enter Data Elements As Per Increasing Order
3
6
8
9
<<====>>
 Hamming Code Is As Shown:
 9 8 6 0 3 1 1
<<====>>
 Place An Error
Enter position to be inverted
If Not Press 0
3
 Hamming Code Sending .  .  .  .  .  .




 Hamming Code Received!!!!!!
9 8 6 0 1 1 1



No Error Detected In Hamming Code
<<====>>
```

b)even parity:

```
Enter No of Data To be Transmitted
3
<<====>>
Enter Parity
1.Odd
2.Even
2
<<====>>
Enter Data Elements As Per Increasing Order
3
5
6
<<====>>
 Hamming Code Is As Shown:
 6 5 1 3 1 0
<<====>>
 Place An Error
Enter position to be inverted
If Not Press 0
1
 Hamming Code Sending . . . . . .



 Hamming Code Received!!!!!!
6 5 1 3 1 1


Error Detected at location 1



Corrected Code is
 6 5 1 3 1 0
```

2)Cyclic Redundancy Check (CRC):

```
Enter dividend: 35

Enter divisor: 12

Message to be transmitted is: 350
```

Conclusion: Thus we have implemented both Hamming Code And CRC using Java Programing.