COMPUTER NETWORKS

EXPERIMENT 3

NAME : GREHA SHAH- 60004200038

BATCH: A2

Aim: - To implement CRC and hamming code as error detection and correction codes.

Theory:

CRC:

The cyclic redundancy check (CRC) is a technique used to detect errors in digital data. As a type of checksum, the CRC produces a fixed-length data set based on the build of a file or larger data set. In terms of its use, CRC is a hash function that detects accidental changes to raw computer data commonly used in digital telecommunications networks and storage devices such as hard disk drives.

CRC is based on binary division and is also called “polynomial code checksum.”

Code:

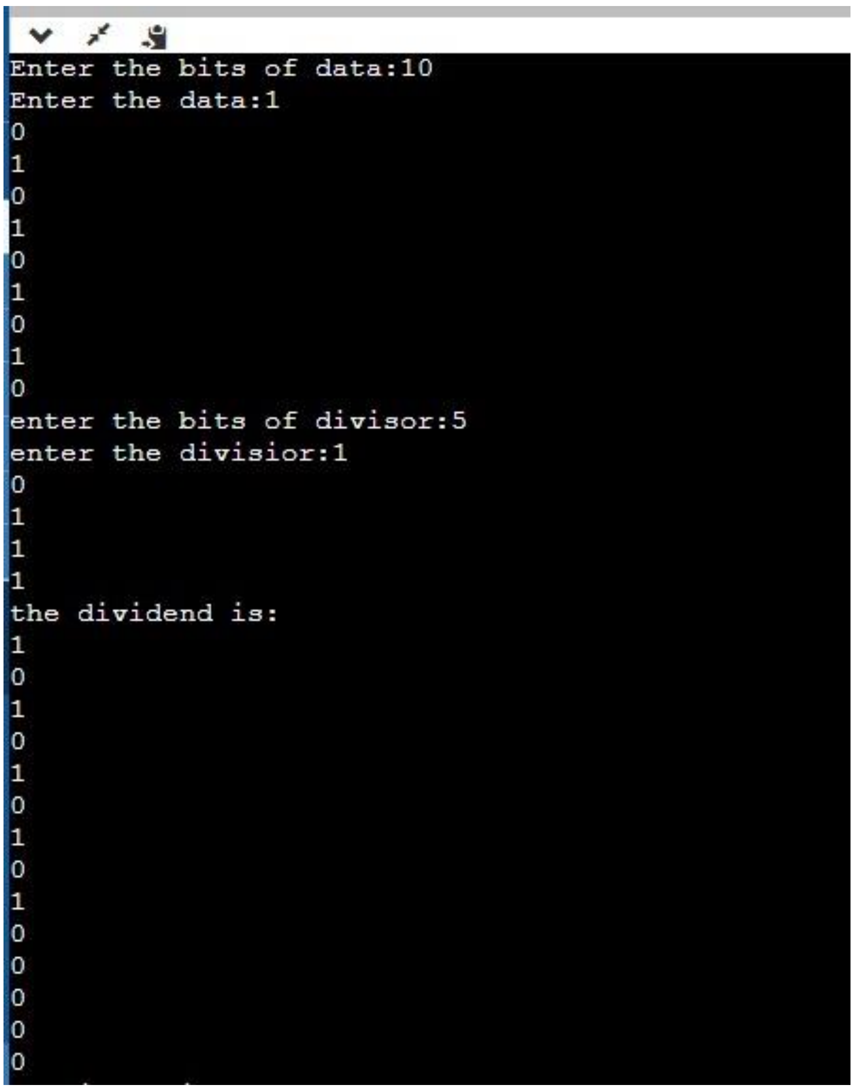
```
#include<stdio.h>

void main()

{

Int n,k,i,j,data[100],divi[100],key[100],rem[100],temp;

    printf("Enter the bits of data:");

    scanf("%d",&n);

    printf("Enter the data:");

    for(i=0;i<n;i++)
```

```
   {
      scanf("%d",&data[i]);
   for(i=0;i<n;i++)
   {
      key[i]=data[i];
   }
   printf("enter the bits of
divisor:");    scanf("%d",&k);
printf("enter the divisior:");
for(i=0;i<k;i++)
   {
      scanf("%d",&divi[i]);
   }
   for(i=n;i<n+k-1;i++)
   {
      data[i]=0;
   }
   printf("the dividend is:\n");
for(i=0;i<n+k-1;i++)
   {
      printf("%d\n",data[i]);
   }
```

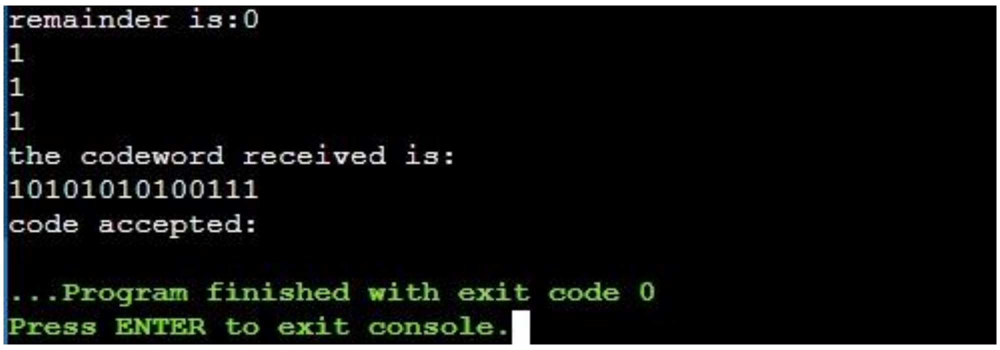
```
      for(i=0;i<n;i++)
    {
        int temp,rem[100];
         temp=i;
if(data[i]==1)
        {
            for(j=0;j<k;j++)
            {
                if(data[temp]==divi[j])
                {


            data[temp]=0;
            rem[j]=0;
                }
         else
                {
                    data[temp]=1;
       rem[j]=1;


                }
                temp+=1;
            }
        }
    }
    printf("remainder is:");
for(i=1;i<k;i++)
    {
        printf("%d\n",rem[i]);
    j=1;     for(i=n;i<n+k-
1;i++)
    {
        key[i]=rem[j];
j++;

    }
    printf("the codeword received is:\n");
for(i=0;i<n+k-1;i++)
    {
        printf("%d",key[i]);
    }
    for(i=0;i<n+k-1;i++)
    {
        temp=i;
if(key[i]==1)
        {
            for(j=0;j<k;j++)
            {
                if(key[temp]==divi[j])
                {
rem[j]=0;
key[temp]=0;

                }
else
                {
rem[j]=1;
key[temp]=1;
                }
                temp+=1;
            }
        }
    }
    int flag=0;
for(i=0;i<k;i++)
```

```
        {
            if(rem[i]==1)
            {
    flag=1;
            }
        }
    if(flag==0)
printf("\ncode accepted:");
        else
        {
            printf("\nnot accepted:");
    /*     for(i=0;i<n;i++)
        {
        printf("%d",key[i]);
        }
        */
    }
```

Output:



```
Enter the bits of data:10
Enter the data:1
0
1
0
1
0
1
0
1
0
enter the bits of divisor:5
enter the divisior:1
0
1
1
1
the dividend is:
1
0
1
0
1
0
1
0
1
0
0
0
0
0
```



```
remainder is:0
1
1
1
the codeword received is:
10101010100111
code accepted:

...Program finished with exit code 0
Press ENTER to exit console.
```

HAMMING CODE:

Hamming code is a block code that is capable of detecting up to two simultaneous bit errors and correcting single-bit errors. It was developed by R.W. Hamming for error correction.

In this coding method, the source encodes the message by inserting redundant bits within the message. These redundant bits are extra bits that are generated and inserted at specific positions in the message itself to enable error detection and correction. When the destination receives this message, it performs recalculations to detect errors and find the bit position that has error.

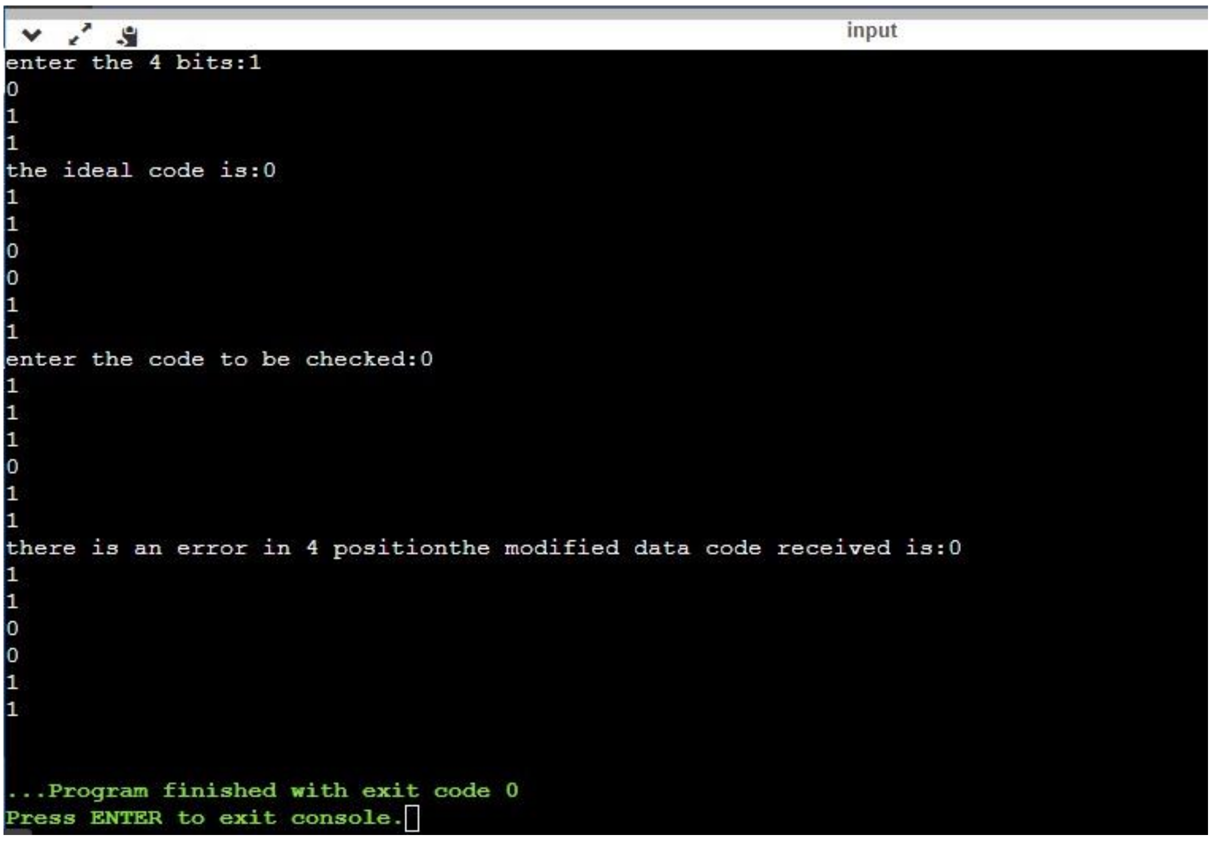
Code:

```
#include<stdio.h>

void main()

{

   int

i,data[100],code[100],p1,p2,p4,p;

printf("enter the 4 bits:");

scanf("%d",&data[2]);

scanf("%d",&data[4]);

scanf("%d",&data[5]);

scanf("%d",&data[6]);

data[0]=data[2]^data[4]^data[6];

data[1]=data[2]^data[5]^data[6];

data[3]=data[4]^data[5]^data[6];
```

```
printf("the ideal code is:");
for(i=0;i<7;i++)
   {
       printf("%d\n",data[i]);
   }
   printf("enter the code to be checked:");
for(i=0;i<7;i++)
   {
       scanf("%d",&code[i]);
   }
   p1=code[0]^code[2]^code[4]^code[6];
p2=code[1]^code[2]^code[5]^code[6];
p4=code[3]^code[4]^code[5]^code[6];
p=p1*1+p2*2+p4*4;
   if(p==0)
   {
       printf("code given is correct:");
   }
   else
   {
       printf("there is an error in %d position",p);
if(code[7-p]==0)
```

```
        {
            code[7-p]=1;
        }
else
        {
            code[7-p]=0;
        }
        printf("the modified data code received
is:");     for(i=0;i<7;i++)
    {
        printf("%d\n",code[i]);

    }
    }
}
```

Output:



```
enter the 4 bits:1
0
1
1
the ideal code is:0
1
1
0
0
1
1
enter the code to be checked:0
1
1
1
0
1
1
there is an error in 4 positionthe modified data code received is:0
1
1
0
0
1
1


...Program finished with exit code 0
Press ENTER to exit console.
```