EXERCISE-4

ERROR DETECTION

AIM:

To write a program for five types of Error detection methods using Python.

SOFTWARE USED:

Python IDLE 3.10

THEORY:

PARITY

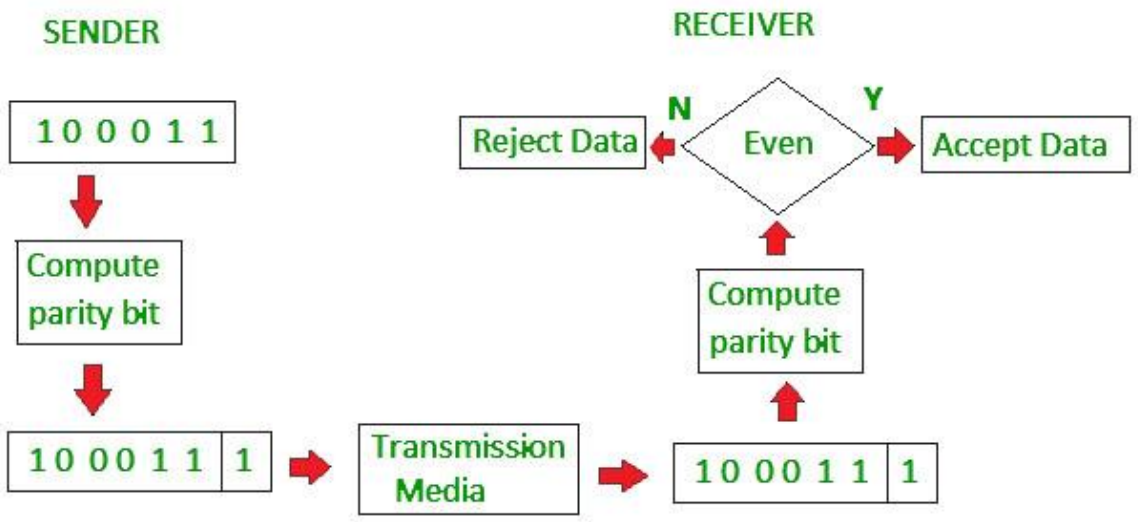
Blocks of data from the source are subjected to a check bit or parity bit generator form, where a parity of :

- 1 is added to the block if it contains odd number of 1's, and
- 0 is added if it contains even number of 1's

This scheme makes the total number of 1's even, that is why it is called even parity checking. Similarly, odd parity can also be made by

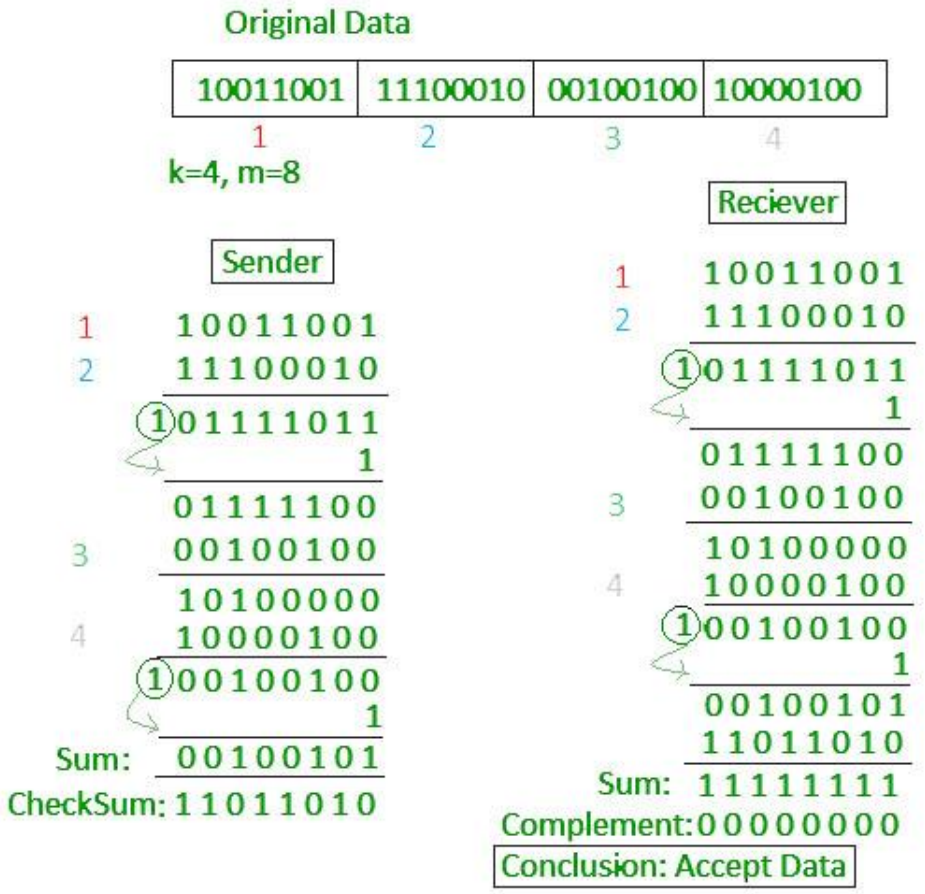
- 1 is added to the block if it contains even number of 1's, and
- 0 is added if it contains odd number of 1's

This scheme makes the total number of 1's odd, that is why it is called odd parity checking.



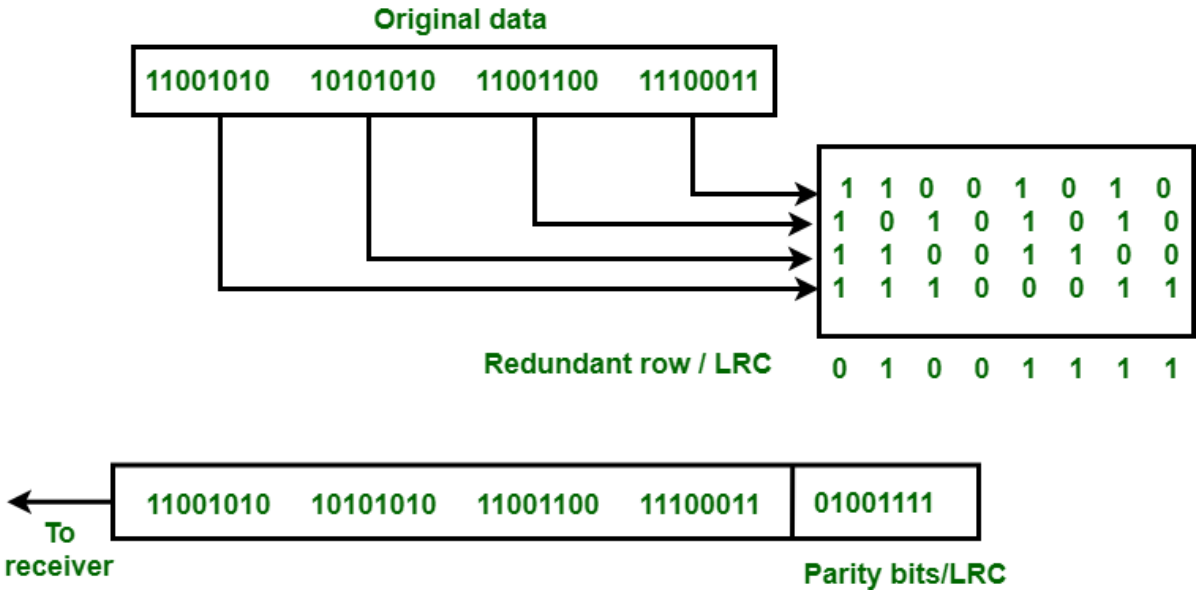
CHECKSUM

- 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.



Longitudinal Redundancy Check (LRC):

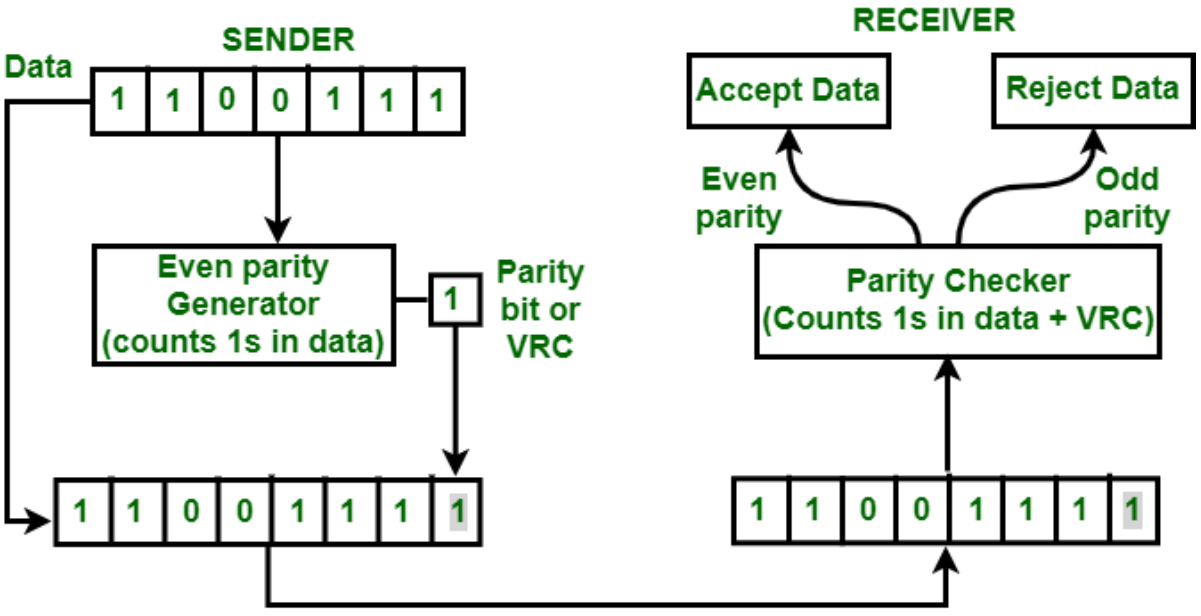
Longitudinal Redundancy Check (LRC) is also known as 2-D parity check. In this method, data which the user want to send is organised into tables of rows and columns. A block of bit is divided into table or matrix of rows and columns. In order to detect an error, a redundant bit is added to the whole block and this block is transmitted to receiver. The receiver uses this redundant row to detect error. After checking the data for errors, receiver accepts the data and discards the redundant row of bits.



Vertical Redundancy Check(VRC):

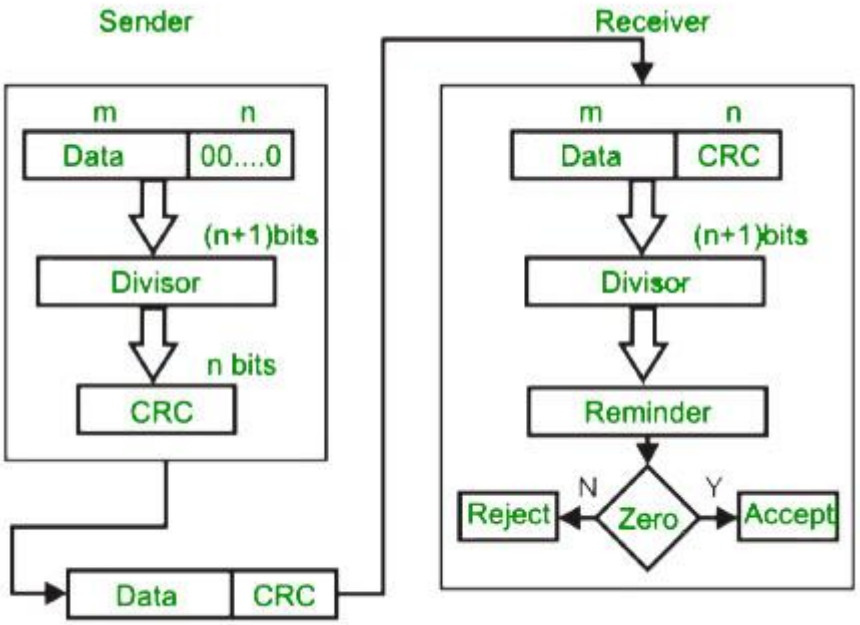
VRC is a redundancy check meant for parallel synchronized bits applied one bit at a time. It uses additional parallel channels for check bits and refers to single-parity bits or larger hamming codes. Vertical redundancy check (VRC) is an error-checking method used on an eight-bit ASCII character. In VRC, a parity bit is attached to each byte of data, which is then tested to determine whether the transmission is correct.

VRC is considered an unreliable error-detection method because it only works if an even number of bits is distorted. A vertical redundancy check is also called a transverse redundancy check when used in combination with other error-controlling codes such as a longitudinal redundancy check.



Cyclic redundancy check (CRC)

- Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.



CODE:

```
def parity():
    print("Parity")
    n=input("Enter the data to be transmitted :")
    c=int(input("1.Even parity\n2.Odd parity\nEnter your choice :"))
    cnt=n.count("1")
    if(c==1):
        if(cnt%2==0):
            print("Parity bit is 0")
```

```
        else:
            print("Parity bit is 1")
    elif(c==2):
        if(cnt%2==0):
            print("Parity bit is 1")
        else:
            print("Parity bit is 0")
    else:
        print("Sorry! You entered wrong choice")


def checksum():
    print("Checksum")
    n=input("Enter the data to be transmitted :")
    l=[]
    for i in range(len(n)//2):
        t=n[(i*2):(i*2)+2]
        l.append(t)
    summ=0
    for i in l:
        summ+=int(i,16)
    b=bin(summ)
    bi=b[2:len(b)]
    bi=bi.replace('1','x')
    bi=bi.replace('0','1')
    bi=bi.replace('x','0')
    d=int(bi,2)+int('1',2)
    h=hex(d)
    bi=bin(d)
    print("Checksum value in hexadecimal is",h[2:len(h)],"\nChecksum value in
binary is",bi[2:len(bi)])


def lrc():
    print("LRC")
    n=input("Enter the data to be transmitted :")
```

```
    l=[]
    for i in range(len(n)//2):
        t=n[(i*2):(i*2)+2]
        l.append(t)
    s=""
    for i in l:
        t=bin(int(i,16))
        if(t.count('1')%2==0):
            s=s+'0'
        else:
            s=s+'1'
    h=hex(int(s,2))
    print("LRC value in hexadecimal is",h[2:len(h)],"\nLRC value in binary
is",s)


def vrc():
    print("VRC")
    n=input("Enter the data to be transmitted :")
    l=[]
    li=[]
    for i in range(len(n)//2):
        t=n[(i*2):(i*2)+2]
        l.append(t)
    for i in l:
        b=bin(int(i,16))
        bi=b.replace("0b","")
        li.append(bi)
    l=[]
    for i in li:
        while(True):
            if(len(i)!=8):
                i='0'+i
            else:
                l.append(i)
```

```
                break
    vrc=[]
    for i in range(8):
        summ=""
        for j in range(len(l)):
            summ=summ+l[j][i]
        if(summ.count('1')%2==0):
            t='0'
            vrc.append(t)
        else:
            t='1'
            vrc.append(t)
    h=hex(int("".join(vrc),2))
    print("VRC value in hexadecimal is",h[2:len(h)],"\nVRC value in binary
is","".join(vrc))


def crc():
    print("CRC")
    dp=input("Enter data polynomial :")
    gp=input("Enter generator polynomial :")
    ngp=len(gp)
    zeros=""
    for i in range(ngp):
        zeros+='0'
    dp+=zeros
    ndp=len(dp)
    idp=dp[0:ngp]
    k=ngp
    global rem
    for i in range(ndp):
        s=bin(int(idp,base=2)^int(gp,base=2))
        ind=s.index('1')
        rem=s[ind::]
        l=ngp-len(rem)
```

```
        kl=k+l
        idp=rem+dp[k:kl]
        k=kl
        if(kl==ndp):
            break
    h=hex(int(rem,2))
    print("CRC value in hexadecimal is",h[2:len(h)],"\nCRC value in binary
is",rem)


while(1):
    ch=int(input("1-Parity\n2-Checksum\n3-LRC\n4-VRC\n5-CRC\n0-EXIT\nEnter
your choice :"))
    print("-"*20)
    if(ch==1):
        parity()
        print("-"*20)
    elif(ch==2):
        checksum()
        print("-"*20)
    elif(ch==3):
        lrc()
        print("-"*20)
    elif(ch==4):
        vrc()
        print("-"*20)
    elif(ch==5):
        crc()
        print("-"*20)
    else:
        print("-"*20)
        break
```

OUTPUT:

PARITY:

```
1-Parity
2-Checksum
3-LRC
4-VRC
5-CRC
0-EXIT
Enter your choice :1
--------------------
Parity
Enter the data to be transmitted :1011010
1.Even parity
2.Odd parity
Enter your choice :1
Parity bit is 0
--------------------
1-Parity
2-Checksum
3-LRC
4-VRC
5-CRC
0-EXIT
Enter your choice :1
--------------------
Parity
Enter the data to be transmitted :1011010
1.Even parity
2.Odd parity
Enter your choice :2
Parity bit is 1
--------------------
```

CHECKSUM:

```
1-Parity
2-Checksum
3-LRC
4-VRC
5-CRC
0-EXIT
Enter your choice :2
--------------------
Checksum
Enter the data to be transmitted :7E8C1604H
Checksum value in hexadecimal is dc
Checksum value in binary is 11011100
--------------------
```

LRC:

```
1-Parity
2-Checksum
3-LRC
4-VRC
5-CRC
0-EXIT
Enter your choice :3
--------------------
LRC
Enter the data to be transmitted :7E8C1604H
LRC value in hexadecimal is 7
LRC value in binary is 0111
--------------------
```

VRC:

```
1-Parity
2-Checksum
3-LRC
4-VRC
5-CRC
0-EXIT
Enter your choice :4
--------------------
VRC
Enter the data to be transmitted :7E8C1604H
VRC value in hexadecimal is e0
VRC value in binary is 11100000
--------------------
```

CRC:

```
1-Parity
2-Checksum
3-LRC
4-VRC
5-CRC
0-EXIT
Enter your choice :5
--------------------
CRC
Enter data polynomial :10100101
Enter generator polynomial :1011
CRC value in hexadecimal is 5
CRC value in binary is 101
--------------------
```

EXIT:

```
1-Parity
2-Checksum
3-LRC
4-VRC
5-CRC
0-EXIT
Enter your choice :0
--------------------
--------------------
>>>
```

RESULT:

Program for five types of Error detection methods (PARITY, CHECKSUM, LRC, VRC, CRC) is written and output is verified.