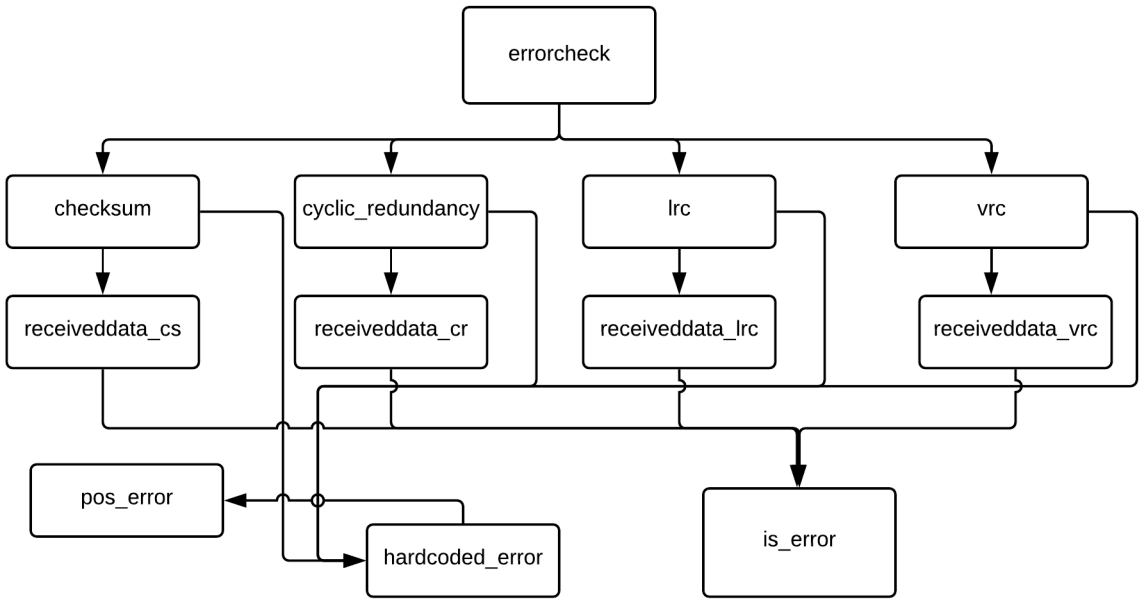
Name: Kushal Kanti Ghosh
Class: BCSE-III
Roll: 28
Section: A1
Assignment: 1
Assignment Deadline: 24/1/2018

Problem Statement: Design and implement an error detection module

Purpose:

During transmission, signals suffer from noise that can introduce errors in the bits travelling from sender to receiver. To make sure whether the receiver's information matches with the sender's information, we use error detection. The main principle is to use error-detecting codes which are additional data (redundancy) added to original data to help us detect error.

Diagram:



Input / Output format:

Input is a text file consisting of 0 and 1.
Output is:
for each error detection technique, for each frame, error is detected or not ("error" or "noerror") .

Code snippet and method description:

```
def generaterror(string):
      #print(string)
      bound=len(string)
      numerror=randint(0,bound//2)                              #number of error bit
      for _ in range(numerror):
            bit=randint(0,bound-1)
            if string[bit]=='1':
                  string=string[:bit]+'0'+string[bit+1:]
            else:
                  string=string[:bit]+'1'+string[bit+1:]
      #print(string)
      #print()
      return string
```

This method introduces error ar random positions of a given codeword by flipping random bits.

```
def pos_error(string,bit):
      if string[bit]=='1':
            string=string[:bit]+'0'+string[bit+1:]
      else:
            string=string[:bit]+'1'+string[bit+1:]
      return string
```

This method injects error at a given poisition.

```
def hardcodederror(string,error):
      for pos in error:
            string=pos_error(string,pos)
      return string
```

This method takes the codeword and a list of positions in which error will be injected as input, and gives the erroneous codeword as output.

Checksum Error detection technique:

```
def wrap(chksumbin,framesize):
	chksumbin=chksumbin.split("b")
	chksumbin=chksumbin[-1]
	while len(chksumbin)>framesize :
			temp=chksumbin[:-framesize]
			chksumbin=chksumbin[-framesize:]
			chksum=int(temp,2)+int(chksumbin,2)
			chksumbin=bin(chksum)[2:]
	return chksumbin
```

Input: checksum (binary string) and framesize. Output: checksum wrapped into framesize

```
def onescomplement(string):
	string=string.replace('0','x')
	string=string.replace('1','0')
	string=string.replace('x','1')
	return string
```

Complement (1's) a binary string.

```
def receivedata_chksum(receivedframes,framesize):
	check=0
	for i in receivedframes:
		#print(i)
		check+=int(i,2)
	check = bin(check)
	check=wrap(check,framesize)
	check=onescomplement(check)
	#print("result : ",check)
	flag=0
	if check.count("0") != len(check):
		#print("error : ",i)
		flag=1
	else:
		a=1
		#print("noerror : ",i)
	if flag==1:
		return 1
	else:
		return 0
```

Receive codewords and check for any type of error.

```
def cs(infile,errorlist):
	framesize=8
	with open(infile,"r") as f1:
		inputstream=f1.read()
	length=len(inputstream)
	inputs=[ inputstream[i:i+framesize] for i in range(0,length,framesize) ]
	outputs=[]
	sendframe=2

	sublist=[inputs[i:i+sendframe] for i in range(0,len(inputs),sendframe)]
	iserror=0
	for senddata in sublist:
		#print(senddata)
		chksum=0
		for i in senddata:
			temp=int(i,2)
			chksum+=temp
		#print("chksum",chksum)
		chksumbin=bin(chksum)[2:]
		#print(chksumbin)
		chksumbin=wrap(chksumbin,framesize)
```

```
            #print(chksumbin)
            chksumbin=onescomplement(chksumbin)
            #print("chksum ",chksumbin)
            errorsend=[]
            for i in senddata:
                  val=0 #randint(0,1)
                  if val==1 :
                        temp=hardcodederror(i,errorlist)
                  else:
                        temp=i
                  errorsend.append(temp)
            errorsend.append(chksumbin)
            iserror+=receivedata_chksum(errorsend,framesize)
      if iserror!=0:
            print("error")
      else:
            print("noerror")
```

Reads the input text file, converts into datawords, converts datawords into codewords, introduces error and calls error checking function.

CRC Error Detection Technique:

```
def encode(dataword,divisor):
      l=len(divisor)
      appendeddata=dataword+"0"*(l-1)
      remainder=division(appendeddata,divisor)
      #print("remainder : ",remainder)
      codeword=dataword+remainder
      return codeword
```

Input: dataword and generator polynomial. Output: codeword

```
def division(dividend,divisor):
      pick=len(divisor)
      tmp=dividend[0:pick]
      while pick<len(dividend):
            if tmp[0]=='1' :
                  tmp=myxor(divisor,tmp)+dividend[pick]
            else:
                  tmp=myxor("0"*pick,tmp)+dividend[pick]
            pick+=1
      if tmp[0]=='1':
            tmp=myxor(divisor,tmp)
      else:
            tmp=myxor("0"*pick,tmp)
      return tmp
```

Divides dividend with divisor.

```
def myxor(a,b):
      l=len(b)
      xor=[]
      for i in range(1,l):
            if a[i]==b[i]:
                  xor.append("0")
            else:
                  xor.append("1")
      xor="".join(xor)
      return xor
```

XORs two binary strings.

```
def receivedata_crc(receivedframes):
      check=0
      flag=0
      for i in receivedframes:
            val=division(i,divisor)
            #print("check val :",val)
```

```
            if val.count("0") != len(val):
                  #print("error : ",i)
                  flag=1
            else:
                  a=1
                  #print("noerror : ",i)
      if flag==1:
            return 1
      else:
            return 0
```

Receive codewords and check for any type of error.

```
divisor="10001001"
def cy(infile,errorlist):
      framesize=8
      divisor="10001001"
      with open(infile,"r") as f1:
            inputstream=f1.read()
      length=len(inputstream)
      inputs=[ inputstream[i:i+framesize] for i in range(0,length,framesize) ]
      outputs=[]


      sublist=inputs[:]
      count=0
      iserror=0
      for senddata in sublist:
            #print(senddata)
            count+=1
            senddata=encode(senddata,divisor)

            errorsend=[]
            val= 0#randint(0,1)
            if count%2==1 :
                  temp=hardcodederror(senddata,errorlist)
            else:
                  temp=senddata
            errorsend.append(temp)
            iserror+=receivedata_crc(errorsend)
      if iserror!=0:
            print("error")
      else:
            print("noerror")
```

Reads the input text file, converts into datawords, converts datawords into codewords, introduces error and calls error checking function.

LRC Error Detection Technique:

```
def receivedata_lrc(receivedframes):
      check=0
      flag=0
      for i in receivedframes:
            val=i.count('1')
            if val%2 == 1:
                  #print("error : ",i)
                  flag=1
            else:
                  a=1
                  #print("noerror : ",i)
      if flag==1:
            return 1
      else:
            return 0
```

Receive codewords and check for any type of error.

```
def lp(infile,errorlist):
```

```
	framesize=8
	with open(infile,"r") as f1:
		inputstream=f1.read()
	length=len(inputstream)
	inputs=[ inputstream[i:i+framesize] for i in range(0,length,framesize) ]
	outputs=[]

	sublist=inputs[:]
	iserror=0
	for senddata in sublist:
		#print(senddata)

		#even pairity
		val=senddata.count('1')
		if val % 2 == 1 : #odd number of 1's
			senddata+="1"
		else:
			senddata+="0"

		errorsend=[]
		val=0 #randint(0,1)
		if val==1 :
			temp=hardcodederror(senddata,errorlist)
		else:
			temp=senddata
		errorsend.append(temp)
		iserror+=receivedata(errorsend)
	if iserror!=0:
		print("error")
	else:
		print("noerror")
```

Reads the input text file, converts into datawords, converts datawords into codewords, introduces error and calls error checking function.

VRC Error Detection Technique:

```
def receivedata_vrc(receivedframes,framesize):
	check=0
	#print("received : ",receivedframes)
	tempdata=[]
	for bit in range(framesize):
		temp=[i[bit] for i in receivedframes]
		temp="".join(temp)
		#print(bit," pos : ",temp)
		tempdata.append(temp)
	flag=0
	for i in tempdata:
		val=i.count('1')
		if val%2 == 1:
			#print("error  ")
			flag=1
		else:
			a=1
			#print("noerror  ")
	if flag==1:
		return 1
	else:
		return 0
```

Receive codewords and check for any type of error.

```
def vp(infile,errorlist):
	framesize=8
	with open(infile,"r") as f1:
		inputstream=f1.read()
	length=len(inputstream)
	inputs=[ inputstream[i:i+framesize] for i in range(0,length,framesize) ]
```

```
	outputs=[]
	sendframe=2

	sublist=[inputs[i:i+sendframe] for i in range(0,len(inputs),sendframe)]
	iserror=0
	for senddata in sublist:
		#print(senddata)

		tempdata=[]
		for bit in range(framesize):
			temp=[i[bit] for i in senddata]
			temp="".join(temp)
			#print(bit," pos : ",temp)
			tempdata.append(temp)
		errorsend=[]
		x=""
		for i in tempdata:
			#pairity
			val=i.count("1")
			if val%2 == 1:
				x+= "1"
			else:
				x+= "0"
		count=0
		for i in senddata:
			val=0 #randint(0,1)
			count+=1
			if val==1 :
				temp=hardcodederror(i,[0,4,7])
			else:
				temp=i
			errorsend.append(temp)
		errorsend.append(x)
		iserror+=receivedata_vrc(errorsend,framesize)
	if iserror!=0:
		print("error")
	else:
		print("noerror")
```

Reads the input text file, converts into datawords, converts datawords into codewords, introduces error and calls error checking function.

Test cases:

1. Error detected by all four schemes: any single bit error
2. Detected by checksum and VRC but not by CRC: flip the bit positions of the 1 bits in the generator polynomial. Since, I have used CRC-7 (x^7+x^3+1,10001001), the 0^{th}, 4^{th} and 7^{th} bit flip cannot be detected by CRC.
3. LRC can detect but VRC cannot: If even number of bits get flipped in a codeword, that cannot be detected by LRC. But VRC can detect it (although in some cases it may fail as well. Like, if even number of codewords get changed at the same positions).