CS425
Programming Assignment No. 1

Due on February 14

General Instructions:

- You can write a program in C, C++, Java or python. (It must take input file name from command line argument)
- You must provide readme file which contains the language you used, command for compile if required, run command and the version of language you used like Python 2.7 or 3.0.

1. CRC checksum

For this program, you will be given a sample input file containing N number of frames in the following format that contains only 0/1's. The framing is done using a byte-oriented framing scheme with flag bytes and byte stuffing. Each frame starts and ends with a FLAG. There is an escape byte ESC which is used to insert "accidental" FLAG byte and ESC byte in the data. If we need to transmit data which has FLAG or ESC byte respectively, then we send it by using ESC followed by (0x20 XOR FLAG) or (0x20 XOR ESC) so that FLAG will never occur in the DATA part. The format of the frame is as follows. Minimum size of data will be 1B (byte).

FLAG = 10101001(®)

ESC = 10100101(¥)

Generator Polynomial = x^7+x+1

FLAG (1B)	DATA	CRC_CHECKBITS (1B)	FLAG (1B)

You will be given two sample input files and corresponding output files of that input. You have to take this filename as a command-line argument of the program and provide the output on the console.

Sample input files contain N (the actual value may differ for different test cases) number of frames containing only 0/1's in a single line without any space (you can read it as a chunk of 8 bits).

Your job is to find the total number of frames which are separated by FLAG, check invalid frames using CRC_CHECKBITS with the given generator polynomial and print the invalid frame numbers (in the order it appears in the input file starting from 1) separated by ',' (if no invalid frame is found leave it blank). If the frame is valid read it and print the character of that ASCII value in a single line.

NOTE: for generating 1 byte of the CRC_CHECKBITS from the generator polynomial, pad one zero at the end. While decoding, ignore the last bit and consider the rest as the CRC checkbits.

Sample output:
12
6,8,10
Cn Assignment

Explanation:
Total number of frames
Invalid frame no separated by **,**   (frame no starts with 1)
Valid data in string format.

Grading Policy

You should not use any external libraries to solve this problem.
If program correctly handles test cases of sample input file (20 points)
If program correctly handles test cases of hidden input file (30 points)

2. Hamming Code

For this problem, you need to decode given binary string(s) to their corresponding characters' paragraph. That paragraph may or may not be correct, which depends on whether you found any incorrect characters in the paragraph. The encoding process employed is as follows :

Given paragraph ⟹ encode each character into 8 bits ⟹ Take each nibble (in order) and encode using Hamming(7,4) ⟹ Append all the 7bit encoded blocks in the same order as they occurred in the original paragraph.

You need to do the reverse of the above process and decode the bit string into actual para.

While decoding the string, you may come across these possible situations:

1^{st} Nibble	2^{nd} Nibble	Char to be used
No error	No error	correct char
No error	Error	Possibly correct char
Error	No error	Possibly correct char
Error	Error	Possibly correct char

There are several key points that you need to keep in mind:

- Char = 8 bits broken into 7+7 bits with hamming encoding of its constituent 4+4 bits
- The source/original paragraph (and not the final decoded) may contain all **ASCII printable characters** (value 32 to 126)
- We have introduced only 1 and 2 bit errors (very few) so as to enable you to detect all the errors present in the bit string. Also, as you might be aware that it is not possible to distinguish 1 bit and 2 bit errors as they both look the same. We can correct all 1 bit errors in Hamming(7,4) successfully but not the 2 bit ones.
- We have mentioned "*possibly correct char*" because it is impossible to know if the error we just corrected was 1 bit or 2 bit.
- The received encoding must contain *#bits in multiples of 14*. If not, output **"INVALID"**, followed by two newlines and the decoding of the following paragraphs (as usual).

INPUT-OUTPUT Format:

Your code will be tested against multiple test cases present in an input file named "input.txt" wherein there will be multiple bit strings (representing a paragraph) with a carriage return followed by a blank line (so finally two carriage returns/newline) in between two paragraphs.
So for input, hardcode that name ("input.txt") in the code itself. For output, you'll print the decoded paragraph with **max error percentage** *(format: [Integer][space]%)* in the next line following the decoded para. After that, insert a new line and continue with the next decoded para, ..so on.

Your output on console would somewhat look like this:

This @s a sample output wi%h di*fer8nt cha123)ters s(^h as : ,./ blah blah u23 hello *&)$_@${}}\@/f!!.?

20 %

This is our second decoded paragraph after two newline/carriage returns.

0 %

INVALID

So on… (this is actually the 3rd paragraph :))

34 %

Example 1:
Original text:
IITK, 1959

Encoded Text (sent) :
1001100001100110011000011001010010110011001001100011001101010100111100010
10100000000010000111101001100001100110011000011010010110000110011001

Received Text (**Sample Input 1**):
1000100001000110011000001001010010010111001001100011001101010100111100111
10100000000010000111101001000001100110011000011010010110000110011010

Sample output 1:
IITK,�195:
60 %

Explanation:
All erroneous characters (i.e. 14 bit blocks) have been shown in color here. Blue represents the first block (/encoded nibble) and the yellow represents the second block of a character.
� corresponds to character with ASCII value 160.
You'll calculate max error percentage as:

((characters received with non zero syndrome) / total characters received) * 100

In the above example, there are a total 10 characters out of which 6 were received faulty, hence 60 % error.

NOTE: *error is calculated character wise and not nibble wise.*
NOTE: *We are referring to it as "max error" because in the worst case, all the characters decoded by Hamming(7,4) decoder could have 2 bit errors and no single bit errors. Therefore, this calculation serves as the upper cap on the error that we may get after final decoding.*

Example 2:
Original text:
#/$.+-CS425cn? Is the, bEzT pswd!#

Encoded Text (sent) :
0101010100001101010101111111010101010011000101010001011001010100110011010
1010101010110011001000011010010110000111000011100110010000110101010100001
10100101110011010000111100110001011010000111111111010101000000000100110000
1100100011111000011010101000000000001111100110011001101110000110011001001
01010101001111000101010000000011001100101010100110001001010001111110110100
1001011001100010101000000000000111100000000001111100001100011110001111100
1101001100010101011010010101010101000011

Received Text (**Sample Input 2**):
0101010100001101110101111101010101010011000110010101011001010100110011011
1010101010110011001000011010010110000111000011100110010000110101010100001
1010010110001000000011110011000101101000011111111101010100000000100110000
1100100011111000011010101000000000000110110001000110010100100001101110011000
01010101001111000101010000000011001100101010100110001001010001111011010 0
1001011001100010101000000000011111000110000011111000011000111100011111100
1101001100010101011011010101010101000011

Sample output 2:
#/$�+-CS425Cn? Is �Pk, bEzT tswd!#
26 %

Explanation:
In the sample output above, � corresponds to characters with ASCII values 190 and 144 respectively. As mentioned already, the original paragraph would only contain “printable ascii characters” (with ASCII value 32-126), but the decoded paragraph may have any 256 characters. *NOTE: You need not worry about the symbols shown on your machine, since your code and our solution code will be run on the same machine and produce the same displayed character.*
As for the percentage error, there were a total of 34 characters, out of which 9 were faulty. Hence (9/34)*100 = 26.47 ≈ 26 % error.
If we observe and compare the original text and the sample output, we’ll find that only 6 characters we have got wrong (in contrast to the max 9). They are { . c t h e p }.

Grading Policy
If the program correctly handles the two given test cases (mentioned in this file) (20 points)
If the program correctly handles test cases of the hidden input file (30 points)

Additional instructions
- All programs submitted will be checked by plagiarism detection software. Honor code policy will be strictly enforced. Write the code by yourself and protect your code.
- Some sample testing files will be provided to you for your convenience. But thorough testing will be done while grading the assignment, so you should test it accordingly before the final submission and not just for the files given.
- For coding in Python, no special libraries required, just the conversions among different data types, etc.