Programming Exercise for Software Developer Candidate: Manchester Encoding Test

Objective

A customer has requested that data be encoded in a particular manner, using Manchester encoding along with some specifics from the Oregon Weather Institute. Your task is to write a program that reads data from the standard input, encodes it according to specification, and writes to the standard output.

Description

There are three steps to how the data should be processed.

Step 1:

The program shall read data from the standard input and process them bit by bit, MSB first. The input may be of arbitrary size.

Step 2:

The Oregon Weather Institute adds, before each bit, an inverted copy of that bit. For each bit of the input, the program shall add an inverted copy of that bit preceding the input bit. That is, a 0 bit shall be encoded as [1, 0] and a 1 bit shall be encoded as [0, 1].

Step 3:

Manchester encoding consists of encoding a 0 as a transition from low to high and a 1 as a transition from high to low. The program shall encode data from Step 2 using Manchester encoding. That is, a 0 bit shall be encoded as [0, 1] and a 1 bit shall be encoded as [1, 0].

Output:

The program shall output the bit stream as 8 - bit characters on the standard output, again MSB first.

The program should not write each bit as a character (that is, it should not output a string of '1' and '0' characters). Each output character should contain 8 bits of the result from Step 3.

The output of the program may contain unprintable characters; this is normal. For example:

```
$ echo -n a | ./encode
```

Figure 1: example console output

Evaluation

The program will be evaluated according to whether it encodes and outputs the data as specified. The source code will also be evaluated for organization, style, and readability.

Notes:

- Please document the method to use in order to compile and run the program.
- It is not requested to display the result of intermediate steps.

Examples

For each of the following examples, it is possible to verify that the output is conforming, by piping the output of the Manchester encoder into `hexdump -C` which will dump the data stream as a sequence of hexadecimal values.

In those examples, we'll consider that the encoder program is named `encode`.

A well behaving decoder shall work on the examples listed there, but it is advised to test the encoder on other kind of data, like actual binary files (images, zip files…) or even better, purely random data as something like `/dev/urandom` can provide.

Example 1

Suppose the input to the encoding program is the character 'a', which we can generate with the command `echo -n a`.

The expected output will look like:

```
echo -n a | ./encode | hexdump -C
00000000  96 69 99 96                                       |.i..|
00000004
```

The bit sequence at each of the steps would look like this:

Step 1: `01100001`

Step 2: `1001011010101001`

Step 3: `10010110011010011001100110010110`

Example 2

Suppose the following input is given to the encoding program:

```
echo 'hello' | ./encode
```

The bit sequence at each of the steps would look like this:

Step 1:

`0110100001100101011011000110110001101111100001010`

Step 2:

```
1001011001101010100101101001100110010110010110101001011001011010100101100101010110101001100110
```

Step 3:

```
1001011001101001011010011001100110010110011010011001011010010110100101100110100101100110100110
0011010011001100101100110100101100110100110011001011001101001011001100110010110100110
01100110010110100101101001
```

Example 3

Please note that your program shall be able to handle any byte values fed into it, not only characters like demonstrated in the previous examples. For example, the value 255 or `0xff` cannot be represented as a character. You can generate it using for example `echo -n -e '\xff'`.

Thus by doing:

```
echo -n -e '\xff' | ./encode
```

You should obtain:

Step 1:

```
11111111
```

Step 2:

```
0101010101010101
```

Step 3:

```
01100110011001100110011001100110
```