Mysterious Parody Bits

Lab 1 - COMP211 - Fall 2020

> "In solving a problem of this sort, the grand thing is to be able to reason backwards. That is a very useful accomplishment, and a very easy one, but people do not practise it much. In the every-day affairs of life it is more useful to reason forwards, and so the other comes to be neglected. There are fifty who can reason synthetically for one who can reason analytically…Let me see if I can make it clearer. Most people, if you describe a train of events to them, will tell you what the result would be. They can put those events together in their minds, and argue from them that something will come to pass. There are few people, however, who, if you told them a result, would be able to evolve from their own inner consciousness what the steps were which led up to that result. This power is what I mean when I talk of reasoning backwards, or analytically." ~ Sherlock Holmes, *A Study in Quiet*

Part 3 of 3. What goes `parity` must come `ytirap`

To resove the final clue of this lab, you will write a program to decode the encoding of `parity.c`. Your source code program should be named `ytirap.c`, of course.

Not only should this program decode the output of `parity.c`, but it should also use the parity bit to detect whether any input bytes are corrupt or improperly encoded.

If a corrupted byte is found, print "Corruption detected!" on its own line and immediately exit with `EXIT_FAILURE`. Before you begin programming, consider the previous part of this lab and convince yourself you know how to detect a corrupt byte that *should be* even-parity encoded.

Example Usage:

```
$ echo "12345" | ./parity | ./ytirap
12345
$ echo "12345" | ./parity
cefij # Expected, properly encoded parity string
$ echo "ceaij" | ./ytirap # Notice an 'a' was snuck in at index 2
12
Corruption detected!
$ echo "cefij" | ./ytirap # Correct parity encoding of 12345
12345
```

The Mystery: Solved

Once your implementation of `ytirap` is working, you can decode the final clue in `clues/03-doughnuts.hex`. Remember, you can output the contents of this file with `cat` before piping it into your decoder(s). It was first encoded with `parity`, and then encoded with `hex`. Once you successfully decode it, answers lie within.

Grading

You must include the course header in all `.c` files. Style is manually graded by course staff, after the late deadline, on the following criteria:

- Use of curly braces around *all* control statement bodies (`if`, `while`, `for`, etc)
- Proper indentation (use the following `vim` trick: `1G=G`)
- All magic numbers are defined as constants
- No global variables
- For parts 0 and 1, only bitwise operations were used in the conversions