Mysterious Parody Bits

Lab 1 - COMP211 - Spring 2025

What truth lies in the center of a zero?

```
      ======================BEGIN TRANSMISSION====================
22537472616E6765206361736520066726F6D207468652073746172742E204120636173652077697
68206120686F6C6520696E20746865206D6964646C652E20200A204120646F7567686E75742E2220
7E2042656E6F697420426C616E630A0A5061727420313A2068747470733A2F2F6269742E6C792F33
516C414872490A
      ====================== END TRANSMISSION ====================
```

In this lab you will follow a crumby trail of nibbles and clues, starting with the encoded transmission above. Bit-by-bit you will shift your way closer to the elusive truth. In the end, you will emerge a bitwiser.

Accept the following GitHub classroom assignment: https://classroom.github.com/a/rnmgGwUv

With your project repository open, click the green "Clone or Download" button. Copy the HTTPS URL. Back in a `learncli` container, clone this url with the `git clone <URL>` subcommand. Replace `<URL>` with your copied URL. (Right click to paste in Windows.) After cloning, change your working directory to the cloned repository.

Part 0 of 3. A hex spell reversed is `xeh`

The encoded transmission is found at the path `clues/00-nibbles.hex` in your repository.

You are given a message that was originally a sequence of ASCII characters. The message was then encoded into hexadecimal digits, with each hex digit stored as an ASCII character byte. Your task is to write a program that reads this hex-encoded data and outputs the original ASCII message.

For example, in the encoded message above, the first two hex digits are `22`. The ASCII character represented by this number is `"`. The next two hex digits are `53`. The ASCII character represented by this number is `S`.

Understanding ASCII and Hexadecimal Conversion

Your program will read each hexadecimal digit one at a time as a `char`. Since the input is a character, your program receives the ASCII value of the digit rather than its numeric value.

Before you begin programming, you should be able to answer the following questions confidently:

1. What is the difference between the ASCII value and the numeric value of a hexadecimal digit? Explain how the ASCII representation of a digit (e.g., `2`) differs from the numeric value it represents (e.g., 2).
2. How can you convert an ASCII `char` into its numeric hexadecimal value? Consider both decimal digits (0–9) and hexadecimal digits (A–F or a–f).

Example: When your program reads the character `2`, its ASCII value is 50. How can you transform this ASCII value into the numeric value 2? What approach would you use for letters such as `A` (which represents 10 in hex)?

Name your C source code file `xeh.c`. Its purpose is to decode the hex message above, or any other message encoded like it. You can add this file to your repository's root directory. **You *must* perform the representation conversions in your own source code using bitwise operators, and may not rely on any library functions.** You can assume the input will contain only an even number of ASCII-encoded hex digits. Notice the hex letters can be either upper or lower case in the encoding. The only other input character you must handle is the new line character, which your program should ignore. Be sure `xeh.c` has the course header lines:

```
// PID: 123456789
// I pledge the COMP211 honor code.
```

Testing

To compile your program to an executable named `xeh`, instead of `a.out`, so you can run it with `./xeh`, use the following `gcc` options (the `-o xeh` option is the one which specifies the output binary name):

```
$ gcc xeh.c -Wall -Wextra -std=c11 -g -o xeh
```

Use the following command to run your program:

```
$ ./xeh
```

Basic Test Cases

Use the following test cases to verify your program's behavior when converting ASCII values to characters:

1. Input: 22 → Output: `"` (Double quotation mark)
2. Input: 53 → Output: `S` (Uppercase letter S)

Create Your Own Test Cases:

3. Lowercase Letter: Provide an ASCII value that corresponds to a lowercase letter and verify that the output is correct.
4. Uppercase Letter: Provide an ASCII value that corresponds to an uppercase letter and verify that the output is correct.
5. Special Character: Provide an ASCII value that corresponds to a special character and verify that the output is correct.

Debugging simple test cases

To debug your program using input from `stdin`, you will need to update your `launch.json` file:

- Comment out the `args` field: This ensures that your program reads input directly from standard input (`stdin`) instead of using predefined arguments.

To see characters print to the console while stepping through the debugger, you must use `fflush()` to force the output to appear immediately:

```
fflush(stdout);
```

Why? By default, output to `stdout` is buffered, meaning it may not appear immediately during debugging. Calling `fflush()` clears the buffer, ensuring you see output as it is produced.

Decoding the Clue

Once your basic tests are working, try decoding the first clue:

```
$ cat clues/00-nibbles.hex | ./xeh
```

Once you've successfully decoded `clues/00-nibbles.hex`, you will now have the clue to bring you to Part 1.

Debugging the Clue

If your program fails to decode the clue, try using the debugger to investigate! Here's how to configure your `launch.json` file to test input from a file:

```
"args": [
    "<",
    "clues/00-nibbles.hex"
]
```

This setup redirects the contents of `clues/00-nibbles.hex` to your program's standard input (`stdin`).

Use breakpoints and step through your code to identify where the issue occurs.

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:

- You must use bitwise operations to combine hex digits.
- Use of curly braces around *all* control statement bodies (`if`, `while`, `for`, etc)
- All magic numbers are defined as constants
- No global variables
- For parts 0 and 1, only bitwise operations were used in the key conversions between hex and ASCII
- No libraries outside of `stdlib.h` and `stdio.h` used.
- Do not commit unnecessary files, such as system files (.DS_Store) or build artifacts (a.out).
- Your commit messages must provide clear and meaningful descriptions of the work being added or changed. A good commit message helps others (and your future self) quickly understand the purpose of a change without needing to dig into the code.