**Homework 1**

**Due date: Thursday, September 17, 2026, 11:59 PM**

**Overview:**

In this assignment, you will write a C program that will act as a UTF analyzer. It will read a single UTF-8-encoded line of text from standard input and print a summary of that text.

**Background:**

UTF-8 is the standard way of storing Unicode text as bytes. Each Unicode character is called a code point and is encoded as a sequence of 1-4 bytes. Most of the standard plain ASCII will be in one byte, while adjacent letters and emojis use 2-4 bytes.

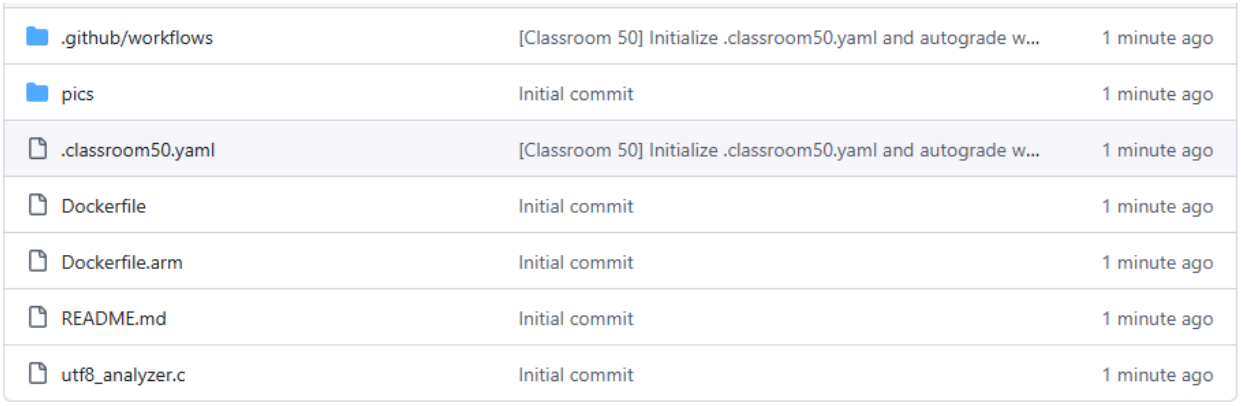
The goal of this assignment is to learn those differences and be able to distinguish between them.

You will practice:

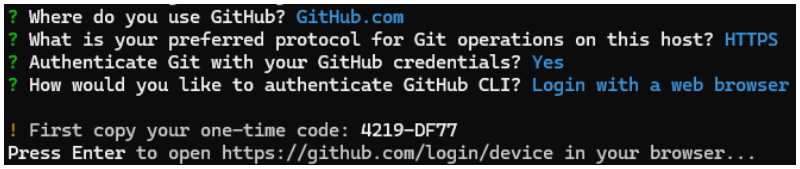
- Working with C strings and raw bytes.
- Understanding how UTF-8 encodes Unicode code points into 1, 2, 3, or 4 bytes.
- Distinguishing between bytes and code points (characters).
- Reading from stdin and writing to stdout.

**Setup:**

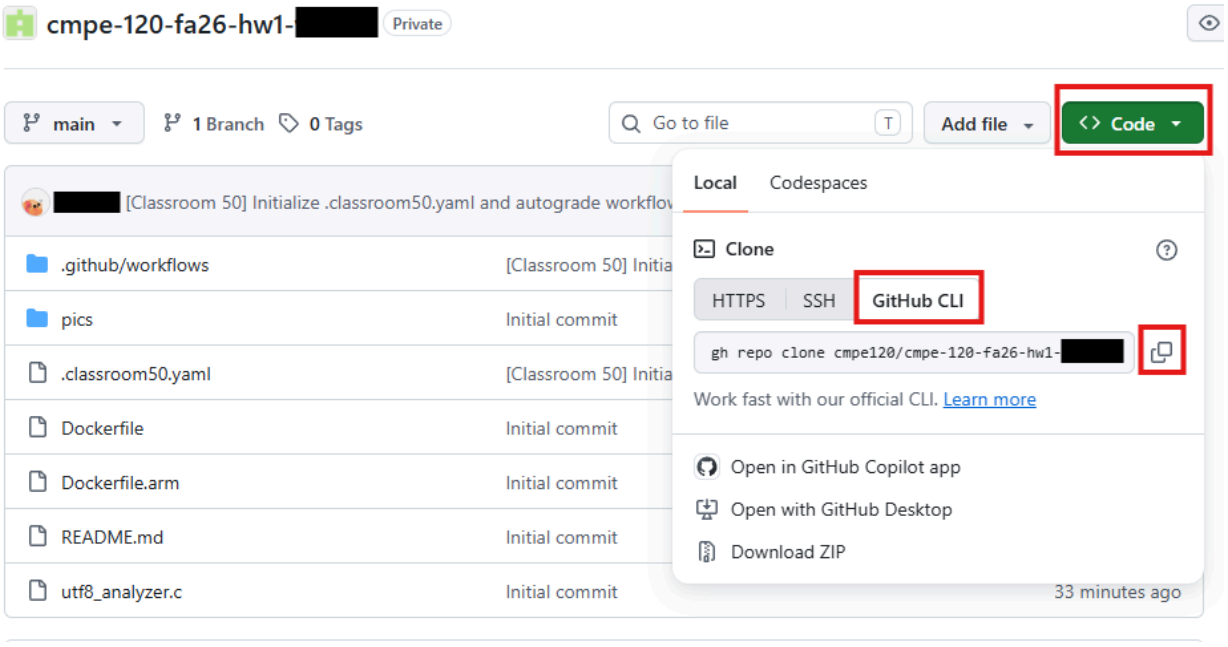
1) Go to Classroom 50 (link: https://classroom50.org) and sign in with your Github account.
2) Click the "Accept Assignment" button on HW1.
    a) Or use this link: https://classroom50.org/cmpe120/cmpe-120-fa26/assignments/hw1/accept
3) Click the "Open Repository" button.
4) A new private repo should be created with the following files in it:



5) Open the repo in your text editor of choice (e.g., VSCode). Then follow these instructions:
    a) Install github cli tool:
        i) On Windows, go to Command Prompt and run:
            (1) `winget install --id GitHub.cli`
        ii) On Mac, make sure you have homebrew installed (if not, install instructions here: https://brew.sh/). Then, go to Terminal and run:
            (1) `brew install gh`
    b) After it installs, in your terminal run:
        i) `gh auth login`
        ii) Answer the authentication questions as follows:




        iii) Your one-time code will be different. Copy it.
        iv) Press enter and follow the instructions in your browser. Enter the one-time code when prompted.
    c) Go to the green code button in the top right corner and change the tab to "Github-CLI"




    d) Copy the CLI command and run it in Command Prompt/Terminal.
    e) Open your repo in your text editor.

6) Follow the instructions in the README.md to set up your Docker environment. Viewing the README on the browser is recommended.

**Troubleshooting:**

- I created my repo but I can't find it
    - Your repo is created under the cmpe120 organization. You can go to classroom50 and there will be a link to it if you lose track of it
- Repository not found/permission denied
    - Make sure you ran `gh auth login`

**Submission:**

You must submit the assignment through Gradescope (via Canvas Assignment "HW1"). When prompted to choose the assignment method, select Github and authenticate with your account where you clone the template repo. Afterwards select the final repository and branch for submission. This repo must have at least the .c file named `utf8_analyzer.c` for proper submission. If the Gradescope is unable to download the repo, then you should download your github repo as a zip and upload that to gradescope instead. Downloading instructions are here:
https://docs.github.com/en/repositories/working-with-files/using-files/downloading-files-from-github

Program Must:

- Read one line of UTF-8 text from standard input (`stdin`).
- Remove any trailing newline (the `\n` that `fgets()` leaves in the buffer). Recommended to use `fgets()`.
- Print the analysis described to standard output (`stdout`).

The input is guaranteed to be valid UTF-8 and no longer than 255 bytes.

**Required Output Format:**

After the input is read, your program must print these 8 lines with the exact labels.

```
Valid ASCII: <true|false>
Uppercased ASCII: <string>
Length in bytes: <integer>
Number of code points: <integer>
Bytes per code point: <space-separated integers>
Substring of the first 6 code points: <string>
Code points as decimal numbers: <space-separated integers>
Animal emojis: <string>
```

**Valid ASCII:** Print true if every byte of the input is a valid ASCII character. False otherwise.

**Uppercased ASCII:** Print the input string converted to its uppercase equivalence. All non-ASCII UTF-8 bytes must be left unchanged, including punctuation.

**Length In Bytes:** Print the number of bytes in the string (not including null terminator).

**Number of Code Points:** Print the number of Unicode Code Points. Ex. 1 ASCII character is one code point. 1 multi-byte UTF-8 sequence is one code point.

**Bytes per Code Point:** For each code point, print how many bytes it occupies (1-4), separated by single spaces

**Substring of the first 6 Code Points:** Print the substring consisting of the first 6 code points of the input (by code point index). If the input has fewer than 6 code points, print the entire input.

**Code points as decimal numbers:** For each code point, print its Unicode scalar value as a decimal integer, separated by single spaces. Ex: 'A' is '65'.

**Animal emojis:** Print every code point that falls inside one of the two animal emoji ranges in the order they appear with no separators between them. If none, print an empty value after the label.

| Range (Hex) | Range (Decimal) | Example Characters |
|---|---|---|
| 0x1F400 – 0x1F43F | 128000 – 128063 | 🐀 … 🐿️ |
| 0x1F980 – 0x1F9AE | 129408 – 129454 | 🦀 … 🦮 |

**Example Input & Output:**

Input:
My 🐩's name is Erdős.

Output:
Valid ASCII: false

Uppercased ASCII: MY 🐩'S NAME IS ERDőS.
Length in bytes: 27
Number of code points: 21
Bytes per code point: 1 1 1 4 3 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1
Substring of the first 6 code points: My 🐩's
Code points as decimal numbers: 77 121 32 128041 8217 115 32 110 97 109 101 32 105 115 32 69 114 100 337 115 46
Animal emojis: 🐩

**Gradescope Results:**

After submission, Gradescope will show a per-test breakdown, out of 100 points:

- 30 points: 6 public test cases (visible immediately).
- 70 points: hidden test cases (not visible to you).

Gradescope checks that you have submitted a `utf8_analyzer.c` file and that your code compiles. Those two checks are not worth points themselves, but if your code fails to compile, you will receive 0 points.