Self implementing printf

Leanna Hakakian

Creative custom extensions

- **reverseString**: reverses an input string
  - I remember we had a hw q where we had to manually do it
- **stringToUpper**: makes input string all capital letters
  - Always use upper() and lower() in python
  - We've been manipulating ASCII a lot
- **intToBinary**: convert unsigned integers to binary strings
  - We had into to hex so I felt int to binary was next obvious option

Most fun part

- Brainstorming creative extensions to implement
- Researching how printf works
- Writing tests

Challenges

- Thinking through edge cases
- Null input cases - what to do with them
- Buffer overflow errors and how to deal with them
- Binary and Hex conversions- became more familiar after studied for the final :)
- ASCII conversions:
    - width = width * 10 + (*input - '0'); // parse the width
        - Subtract ‘0’ from numeric char gives numeric val
        - Mult by 10 to shift the digits left base 10

Challenges continued

Learning and implementing va_args: variable arguments feature in C

- Allows functions to accept an undefined number of arguments
- **va_list**: A type to hold information about the variable arguments.
- **va_start**: Initializes a va_list to retrieve arguments.
- **va_arg**: Retrieves the next argument from the list.
- **va_end**: Cleans up the va_list when done.

What have I learned and next steps

- “Modular code”: learned and implemented it
- Really feel confident with C programming skills
  - Love switch case in particular
- Need to do vigorous testing
- If I had more time and had to redo this project:
  - Add more modifiers
  - Implement more parameters
  - Missed some comments

Code walkthrough notes

- Most challenging part: buffer overflow understanding
- Tools used:
  - `#include <string.h> library`
    - `Strcmp, strlen, strncpy`
  - Define limits for testing
  - Defining error codes
- Setting up the IDE