CIS*2500

Style Guidelines

Compile code using (**without error or warning**): `gcc –std=c99 –Wall`

Identifiers:

- Variable and function names are lower camelCase
- Type names are upper CamelCase
- Single letter identifiers may only be used for loop counters

Variables:

- All variables are declared at the top of their block, in the smallest scope possible
- Variables are declared and initialized on separate lines.
- No global, or static variables are allowed.

Comments:

- /*ansi C style*/ comments must be used for function comments, and multiline comments
- Ansi C style comments, or //c++ comments can be used elsewhere
- Every function must have a function comment above it's declaration

Functions:

- Main must always return an int
- Functions must always verify their actual parameter's validity
- Function comment structure:

```
/**
 * functionName
 * A short description of what the function does
 * IN: parameters, and what they represent
 * OUT: return value, and what it represents
 * POST: Side effects of the function being run
 * ERROR: Error cases
 */
* IN, OUT, POST, ERROR - (if any)
```

Modularity:

- Main must be in it's own .c file
- Functions should be grouped in files by type (ie: all file io in one file, all UI in another, .etc)
- Each non-main .c file must have a corresponding .h file
- .h files contain function prototypes, struct declarations, typedefs and constants
- .h files always use include guards (recommended format: __USERNAME_FILENAME__)
- Function prototypes must be in the same order that they are defined in the .c file

Structure:

- Each assignment folder must consist of a set of subfolders:
  - src/ <- Contains all .c files
  - include/ <- Contains all .h files
  - doc/ <- Contains all documentation
  - assets/ <- Contains any files that the program needs to run (input or output)

Readme:

- Named “README"
- 5 sections inside file:
  - "Compilation", "Usage", "Testing", "Known limitations", and "References"

Makefiles (Not required until covered in lecture):

- Named “Makefile"
- First target is "all", with a list of dependencies, and no commands
- Upon completion, "all" builds all required executables