COSC 3750 --- Homework: Simple `ls`

Total Points: 50

In this assignment you will implement a simplified version of the Unix `ls` command in Go. This version does **not** support flags yet. Your goal is to correctly list files and directories, apply basic coloring when appropriate, and replicate key ordering behavior of real `ls`.

Real `ls` put things in columns when outputting to the terminal. The algorithm behind it is really convoluted, so just put everything on its own line.

This assignment focuses on:

- Filesystem inspection
- Correct use of `os.Lstat`
- Sorting behavior
- Detecting terminal vs piped output
- Understanding file mode bits

Learning Goals

By completing this assignment you should be able to:

- Use `os.Lstat` correctly
- Distinguish between files and directories
- Sort targets and directory entries lexicographically
- Detect whether stdout is connected to a terminal
- Use file mode bitmasks (e.g., `0111`) to detect executables
- Write output using streams (`io.Writer`)
- Reproduce core Unix tool behavior without shortcuts

Program Requirements

Your program must compile to:

```
gols
```

Compile using:

```
go build -o gols
```

Run using:

```
./gols [FILE...]
```

Behavior Specification

1. No Flags (This Assignment Only)

You are implementing only the base behavior of `ls`.

There are **no flags** in this assignment.

---

2. Argument Handling

If no arguments are provided:

```
./gols
```

You must list the contents of the current directory (`.`).

If arguments are provided:

```
./gols file1 dir1 file2 dir2
```

You must:

1. Partition targets into:
    - Files
    - Directories
2. Sort each group lexicographically.
3. Print:
    - All files first (sorted)
    - Then directories (sorted)

---

3. File Targets

If a target is a file:

- Print only its base name.
- One per line.

- Do not print a directory header.

Example:

```
./gols fileA fileB
```

Output:

```
fileA
fileB
```

---

4. Directory Targets

If a target is a directory:

- List its contents.
- Sort entries lexicographically by name.
- Print one entry per line.
- Skip hidden files (names beginning with `.`).

Multiple Directories

If more than one directory target exists, print a header before each directory listing:

```
dirname:
```

Separate directory listings with a blank line.

Example:

```
./gols dir1 dir2
```

Output:

```
dir1:
fileA
fileB

dir2:
fileC
fileD
```

5. Color Output

Your program must apply color **only when writing to a terminal**.

You can use the included `IsTerminal()` function, but read through it!

Color Rules

When outputting to a terminal:

File Type	Color
Directory	Blue
Executable regular file	Green
All others	Default color

When output is redirected or piped:

- No color codes should be printed.

Example:

```
./gols
./gols | cat
./gols > out.txt
```

Only the first should contain ANSI escape codes.

If I were you, i would check the GitHub examples that just so happen to show how to print colors and have blue/green defined

6. File Type Detection

You must use:

```
os.Lstat(path)
```

```
info, err := os.Lstat(path)
mode := info.Mode()
```

Do **not** use `os.Stat`.

Detect:

- Directory → `info.IsDir()`
- Executable → `mode.IsRegular() && (mode & 0111) != 0`

`stat` vs `lstat` (quick explanation)

- `stat(path)` follows symlinks and returns info about the **target** file/directory.
- `lstat(path)` does **not** follow symlinks and returns info about the **link itself**.

Why this matters

- If `link -> real_file`:
    - `stat("link")` reports metadata for `real_file`
    - `lstat("link")` reports metadata for `link`

When to use each

- Use `stat` when you want to treat symlinks like the thing they point to.
- Use `lstat` when you need accurate file-type inspection (including detecting symlinks) without dereferencing.

---

7. Sorting Requirements

You must:

- Sort file targets lexicographically
- Sort directory targets lexicographically
- Sort entries inside each directory lexicographically
- Note for top level entries it goes files then directories
    - `ls directory . file.out` Will produce:

```
file.out

.:
files
and
dirs
lexographically

directory:
files
and
dirs
lexographically
```

- Meaning:
    - Arguments supplied should be sorted into sorted files, then sorted directories
    - Entries within a directory are sorted lexicographically, but files and directories are interspersed

> Lexicographically refers to ordering sequences (like strings or lists) based on alphabetical or dictionary rules, rather than numerical value. It compares items element-by-element from left to right; for instance, "apple" comes before "apply" because 'e' comes before 'y', even if "apply" is shorter. This method is crucial in computer science for sorting, algorithm optimization, and managing structured data sets, extending to ordered sets beyond just letters.

Do not rely on filesystem order.

---

8. Error Handling

If a target cannot be accessed:

- Print an error to **stderr**
- Continue processing remaining targets

Example:

```
gols: cannot access 'missing.txt': no such file or directory
```

Constraints

You MUST

- Use `os.Lstat`
- Use `filepath.Join` when listing directory entries
- Sort targets and entries explicitly
- Write to an `io.Writer`
- Detect terminal output before applying colors

You MAY

- Use `os.ReadDir()` to get the directory contents
- Include the `sort` package, but it may not be helpful in some cases

You CANNOT

- Use `exec.Command("ls")`
- Use `os.ReadFile` or load directories manually
- Assume UTF-8 characters are single bytes
- Use third-party packages
- Use `fmt.Print*` you must use a writer to write to the output
  - You may use `fmf.Fprintf` to write to `os.stderr`

Suggested Structure

You are encouraged to organize your code as follows:

```
gols.go
/fucntions/simplels.go
/functions/color.go
/functions/isTerminal.go
/functions/dirFilter.go
```

Suggested functions:

```
func SimpleLS(w io.Writer, args []string, useColor bool) //A simple ls for when no flags provided
func IsTerminal(f *os.File) bool //checks if outputting to a terminal or being piped/redirected
func (c color) ColorPrint(w io.Writer, s string) //Prints appropriate entries in color, or colorless if regular file
func dirFilter(entries []os.DirEntry) []os.DirEntry //removes any hidden files from the dir listing, note not exported
```

Grading (50 Points)

Category	**Points**
Correct argument partitioning (files first, dirs second)	10
Correct sorting (targets + directory entries)	10
Correct directory listing behavior	5
Correct color behavior (TTY only)	5
Correct executable detection	5
Proper use of Lstat and filepath.Join	5
Error handling	5
Proper Submission/README completion	5
Total	**50**

Programs that do not compile or crash without completing will receive an automatic 50% deduction.

Testing Suggestions

Test:

```
./gols
./gols file.txt
```

```
./gols dir1 dir2
./gols file.txt dir1 file.txt
./gols > out.txt
```

Create:

- A directory
- An executable file (`chmod +x`) (good news, you should be making one when you compile!)
- A regular file
- A hidden file (`.hidden`)

As with gocat you should use the *real* `ls` and compare your results to it*

*** other than `ls` putting things in columns because again, its convoluted so don't bother**
