Project 2 - Text Assistants

Objectives

- Familiarize yourself with manipulating C strings
- Practice with array operations, like indexing and shifting
- Practice using `fgets` and `getline` libraries to read from Standard In and write to Standard Out.
- Experience using small programs chained together in the shell

Introduction

Unix/Linux style C programming often utilizes small programs with simple purposes strung together to solve complicated problems.

Suppose we have two protagonists, Jen and Carlos. Jen and Carlos both have been hired by a local library to edit scans of some old books. The books are so old, the scanning software often incorrectly identifies spaces, capitalization, and more. The Librarian wants the scanned text correctly capitalized, with no duplicate spaces, and the width of each line to be limited to 40 characters (including newline).

The library expects this to be a manual process, and Jen and Carlos are told it will take them one week of work for each scan. Little does the Librarian know: **Jen and Carlos know C!**

Our protagonists not only complete the work for one scan in no time, but they claim they can do any number of others in just a few seconds each! How did they do it?

The Solution

Our protagonists used their knowledge of C and shell scripting to create six programs that each partially solve the Librarian's problem. Each program modifies its input in only one way, line by line. The programs are then combined using pipes to fix the errors in the text. Each program reads from standard input and writes to standard output. Here is a high level description of each program.

`shout`	- capitalize all alphabetic characters in the input
`whisper`	- lowercase all alphabetic characters in the input
`jn`	- replace newlines with spaces
`wrp <num>`	- wrap text to a newline if it exceeds a column limit
`cap`	- properly capitalize characters occurring after `.?!`
`dedup <char>`	- removes repeat occurrences of char

Starter Files

- README.md
- Makefile
- shout.c
- whisper.c
- wrp.c
- jn.c
- cap.c
- dedup.c
- scan.jpg
- scan.txt
- rubric.txt

Tasks

- **Getting started**. Copy the files from the CS253-resources repository, and add them to your portfolio. You can do this by using the following copy commands in the terminal, if you have both repositories cloned:

  ```
  cd <parent folder>/CS253-resources
  git pull
  cp -r projects/p2/* <portfolio>/p2/
  ```

  Where <parent folder> is the location of the class repo, and <portfolio> is the location of your class portfolio. **Git pull is important!**

  Each of the six C programs don't work. *shout*, *whisper* and *jn* use the function `fgets`. **Be sure to read the manual page for `fgets`**. Start by completing *shout, whisper* and *wrp* followed by *jn*, *cap* and *dedup*.

- **`shout.c` and `whisper.c`**. These programs modify the input by replacing alphabetic characters with their upper-case or lower-case equivalents. `ctype.h` contains functions that will be useful for detecting if a character is lower or upper case. *Alternatively*, you can do this by bitsetting the 6th bit of a character to either 1 or 0 (cool!).

  ```
  $ shout
  My input shown here
  MY INPUT SHOWN HERE
  $ whisper
  CAPS LOCK DOESN'T WORK ANYMORE!!
  caps lock doesn't work anymore!!
  ```

- **`jn.c` and `wrp.c`**. *jn* is a program for replacing newlines with a space. *wrp* is a program for limiting the length of a line to a certain number of columns. Instead of using `fgets` for *wrp*, Jen and Carlos wanted to use another popular input reading function, `getline`. Read the man page for `getline`, which includes an example of using it. **You must use `getline`**!

```
$ echo "multiple
 lines" | jn
multiple lines
$ wrp 10
Each line. Can only be 10 chars long. Cool.
Each line.
Can only
be 10
chars
long.
Cool.
$ wrp 10 | jn
In this program, the lines are split by wrp and
joined.
In this program, the lines are split by wrp and
joined.
```

Hints: `wrp` limits the distance between newline characters in a string to at most *num*. If a word is longer than *num*, keep the word intact. Draw pictures of the strings!

```
$ wrp 3
An apple a day...
An
apple
a
day...
```

- **`cap.c`**. This program's function is to correctly capitalize its input. Input is correctly capitalized when the first letter of any new sentence is upper case. Only consider . (period) ! (exclamation point) and ? (question mark) appropriate ends to a sentence. Again, use the `getline` library.

```
$ cap
my sentence needs help! it's difficult
My sentence needs help! It's difficult
to read like this. who
to read like this. Who
can help? you?
can help? You?
```

Hint: The first character of the first line is always capitalized. The first character of any other lines are not, unless they come right after punctuation on a previous line.

- **`dedup.c`**. This program is the most fancy that Jen and Carlos wrote. In translating the image, the scanning software often entered multiple spaces between words and after sentences. They both thought it was more versatile to create a program that takes an argument to be the character that is deduplicated. As such, `dedup` takes the first character of its first argument to be the character that is deduplicated. If no argument is provided, output a usage message and exit with a non-zero value.

```
$ dedup
Usage: dedup <char to dedup>
```

When a character is used as the argument, `dedup` uses it when scanning the input to detect multiple occurrences of that character. When there are multiple occurrences, each successive occurrence is removed from the output.

```
$ dedup l
helllo
helo
```

```
$ dedup " "
Some  sentences start   after two spaces.  Like this.
Some sentences start after two spaces. Like this.
```

- **Combining the programs**. Jen and Carlos used the shell to combine the utility of each program. They used **file redirection** to read `scan.txt` as standard input:

  ```
  $ whisper <scan.txt
  ```

  They used **pipes** to connect the programs together:

```
$ whisper <scan.txt | jn | dedup " " | cap | wrp 40 > expected-out.txt
```

  Finally, they used file redirection to output the reformatted text to an output file for the Librarian:

```
whisper <scan.txt | jn | dedup " " | cap | wrp 40 > expected-out.txt >expected-out.txt
```

- **Additional Requirement**. Do not use the functions in string.h. If you need functions that help with each problem, build your own.

Testing

Begin by testing each program manually as you write it. `shout`, `whisper`, and `jn` are the simplest to test. Since they read from standard input, call the program under test and type some input followed by **control-d** (EOF).

```
$ shout
i like to yell when I write messages
I LIKE TO YELL WHEN I WRITE MESSAGES
Ctrl-d
$
```

Do this with each program to ensure it functions as you expect. To send a space to dedup, call it like:

```
    $ dedup " "
```

To send a newline to dedup, call it like:

```
    $ dedup $'\n'
```

Of course, we cannot forget about the Librarian's problem! The assignment includes a scan (scan.jpg) and the scanning software's attempt at recognizing some of the text.

Jen and Carlos used the following command to produce the output found in `expected-out.txt`.

```
$ whisper <scan.txt | jn | dedup " " | cap | wrp 40
```

To reproduce their results, run the same but redirect it to your own output file:

```
whisper <scan.txt | jn | dedup " " | cap | wrp 40 > myout.txt
```

Then, use the diff program to determine if your output matches our protagonists. No output means it matches perfectly!

```
    $ diff expected-out.txt myout.txt
```

Documentation

Each C program should include proper documentation about its function and usage. The more complicated programs (wrp, dedup and cap) should definitely have inline comments that help explain how you solved the problem.
Also, complete a README which describes the solution and your experience solving it.

Submission

Submit the completed programs in your portfolio, under directory p2. Take care to not include any of the compiled programs.

Bonus (Optional)

Linux distributions include a text dictionary (/usr/share/dict/words). How well do your programs work processing those?

```
$ shout </usr/share/dict/words
```

This is very slow as it is spending all of its time writing to the console. Stop it by sending the signal SIGSTOP with ctrl-c. Redirect it to nothing for a performance measurement!

```
$ time shout </usr/share/dict/words > /dev/null
real    0m0.084s
user    0m0.072s
sys     0m0.007s
```

That file contains nearly 500,000 words, one per line. 84ms is pretty fast! Include your times in your README.