Basic - Regex

EECS 201 Fall 2023

Submission Instructions

This assignment will be submitted as a repository on the EECS GitLab server. Create a private, blank, **README-less (uncheck that box!)** Project on it with the name/path/URL `eecs201-basic-regex` and add `brng` as a Reporter. The submission branch will be `main`. If this branch is not already the default initial branch, you initialize the local repo with an additional argument: `git init --initial-branch=main` if your version of Git is recent enough. Otherwise you can create a branch with this name after your first commit. The repository should have the following directory structure, starting from the repository's root:

```
/
|-- report.txt
|-- grep/
| |-- cap-vow.sh
| |-- ing.sh
| |-- n-letter.sh
| |-- same-lower-vowel.sh
|
|-- sed/
    |-- c89ify.sh
```

Preface

In this assignment you'll be provided yet another zipped archive containing some starter empty files and scripts. Use your preferred tool to retrieve this file and extract it (see Basic - Git 1 if you need a review).

`https://www.eecs.umich.edu/courses/eecs201/fa2023/files/assignments/basic-regex.tar.gz`

Initialize a Git repository in the extracted `basic-regex` directory as per the submission instructions.

1 Regex fun

As mentioned in lecture, `grep` is a utility that finds patterns in files. These patterns are by default POSIX basic regular expressions; `egrep` or `grep` with the `-E` flag will interpret patterns as POSIX extended regular expressions. For this question we'll be looking at an American English dictionary.
This file is located under `/usr/share/dict/american-english`. If you don't have it, on Ubuntu (and WSL Ubuntu) you can get it via the `wamerican` package and on Arch you can get it via the `words` package. If you are on macOS, you probably already have a dictionary file `/usr/share/dict/words` which you can use for the following examples and as input for this part of the assginment. `/usr/share/dict/words` is a standard file that contains a list of dictionary words; it's probably symlinked to an appropriate dictionary file.
If you want to, you could also grab the copy on the course server.

```
$ scp youruniqname@peritia.eecs.umich.edu:/usr/share/dict/american-english .
```

`scp` is a utility that can copy files between one computer and another (when the remote computer is addressable by the local computer). In particular, this one will copy the file on the server to the current directory. You'll have to change the invocation of the scripts to refer to this particular file path. Do note that these scripts can work with any file that has a word on each line in it: you can take advantage of this to do your own simple testing, just replace the dictionary path argument with the path to your dictionary file.

1. Feel free to use `/usr/share/dict/american-english` or `/usr/share/dict/words`.
2. `cd` into the `grep` directory.

3. Run `$ grep "world" /usr/share/dict/american-english`. This finds and prints out words in the dictionary that contain "world" anywhere in the word.
   (If your dictionary file is somewhere else, use that file path instead e.g.
   `$ grep "world" american-english` if it's in your current directory.)

4. Run `$ grep "^[A-Z]" /usr/share/dict/american-english`. This finds and prints out words in the dictionary that start with a capital letter.
   (If your dictionary file is somewhere else, use that file path instead e.g.
   `$ grep "^[A-Z]" american-english` if it's in your current directory.)

Now onto the question proper:

1. Implement the functionality described in each of the Bash scripts. You may only use one `grep` command in each script. Each script takes in the path to some dictionary file as an argument. This can be the American English dictionary or your own test dictionary (a file with a word on each line). Feel free to use ERE via the `-E` flag for `grep`.

2. Remember that the path to the dictionary is an argument for the script, and that you should pass that argument to `grep`! **Do not hardcode what dictionary `grep` uses!** This is determined at runtime by the person (or autograder) running the script!

3. Stage and commit your changes.

2 Searching and replacing text with `sed`

`sed` is a utility that is able to perform pattern searches and replacements. By default `sed` will use POSIX BRE unless a parameter is specified to use ERE.

1. `cd` into the `sed` directory.

2. Run `$ sed -e 's/hello/world/' <<<'hello user hello`. What `sed` did is replace the first instance of "hello" with "world" of each line of input. `s` is the command to "substitute" texts, with the following charactering serving as a delimiter (`sed -e 's@hello@world@'` would also work, with it using '@' as a delimiter instead of '/'). The `s` command is of the format `s/pattern/replacement/flags`.

3. Run `$ sed -e 's/hello/world/g' <<<'hello user hello`. Note that both "hello"s were replaced. The `g` at the end of the `sed` `s` command is a flag that says to replace all matches, not only the first.

4. Run `$ grep -E '(hello) (.*) \1' <<<'hello user hello'`.
   Try it again with a here string of `'hello user goodbye'`. Note how we can re-use part of a pattern using parentheses and a backreference.

5. Run `$ sed -E -e 's/(hello) (.*) \1/world \2 world/g' <<<'hello user hello'` Note the `-E` flag: this puts `sed` into ERE mode. Note the use of backreferences in both the pattern and replacement fields.

Now onto the question proper. Take a look at `saxpy.c`. Note how it uses `//` for comments.

1. Run `$ gcc -pedantic -Werror -std=c99 saxpy.c`. This will compile the C program following the C99 standard. It should compile successfully. Feel free to run the `a.out` binary that is produced.

2. Now run `$ gcc -pedantic -Werror -std=c89 saxpy.c`. This will compile the C program following the C89 ("ANSI C") standard. It should fail to compile. That is because C++ actually introduced the use of `//` for single single line comments, support for which the C99 standard added to the C language.

3. In the provided `c89ify.sh` file, implement the script as described by its usage printout. What this script is supposed to do is take in a list of C files **as arguments** and then for each C file, replace the `// comment` comments with `/* comment */` style comments and put a "fixed" version of the file into a new file that tacks on `.c89` before the `.c` extension. For example, `./c89ify.sh saxpy.c daxpy.c` will produce fixed versions in `saxpy.c89.c` and `daxpy.c89.c` and leaving `saxpy.c` and `daxpy.c` as they originally were.

4. Some notes

   (a) Remember you can test out `sed` at the command line to quickly iterate on your `s` command: by default `sed` is non-destructive and simply outputs the transformed text (and if you need to save it to a file, remember that output redirection is a thing!).

   (b) `sed` is able to read from a file as an argument or as redirected input e.g. `sed 's//' file` or `sed 's//' < file`.

   (c) Try using a backreference!

   (d) Keep in mind `sed`'s `s` command will replace what got matched as a whole: try matching against the whole comment, slashes and all, then replacing it with the correct kind of comment.

5. To test if the replacement worked, try compiling the files with the `-std=c89` flag. (By the way, deleting the comments aren't a solution; we can check to see if the actual comment messages are still there :p)

6. Stage and commit the finished `c89ify.sh` script.

3 Conclusion

1. Add and commit any changes you intend to submit.

2. Create a file called `report.txt`.

3. On the first line provide an integer time in minutes of how long it took for you to complete this assignment.

4. On the second line and beyond, write down what you learned while doing this assignment. If you already knew how to do all of this, put down "N/A".

5. Add and commit this `report.txt` file.