CPSC 223C: C Programming - Spring 2020 Project One, **grep from ed (say “e d”, not “ed”),** due Wednesday, 25 Mar 2020

Based on Princeton’s CS333 course assignment: http://www.cs.princeton.edu/courses/archive/spr08/cos333/ed_to_grep.html

Source code for ed.c is provided on Titanium.

As you know, Ken Thompson and Dennis Ritchie co-created Unix. You may not know that Ken Thompson created the original grep command in one evening, beginning with the source code for Unix's ed editor. (Source code for ed.c is included in this assignment.)

Anything Ken Thompson did, surely we at CSUF can also do! After all, we have the following advantages:

Advantages:
(1) We have two weeks (not one evening) in which to complete it
(2) We have have many examples of what grep does and how it is used
(3) ed is now written in C. At the time Thompson wrote grep, it was written in PDP-11 assembler.

Disadvantage?:
We are not Ken Thompson, co-creator of Unix, who now works at Google. (but one day, who knows?)

The source code for `ed` (`ed.c`) is approximately 1700 lines long **(code shown on Titanium)**, and comes from the 1989 version of `ed`. (It is written in a style that is typical for mid-1970's Unix code: concise, efficient, and basically uncommented. In other words, much in the same style as employers expect developers to write their code (well, maybe not the uncommented part).

Your `grep` program should use the code for regular expression processing. You will have to throw away quite a bit of code, since your version of `grep` should not need more than about 400 lines. Will you need to add your own code? Yes, but not more than about 30 lines or so.

Your `grep` code should be able to read its input from stdin or from one or more named files, like so: `grep regexpr [files...]`

If there is more than one file to search, each matching line should be prefixed by the filename it came from (e.g., file1: I found this line file2: and this line file2: and this line too)

You don't have to provide any of the `grep` options (`-i -n -v ...`), but must return status values, such as:
0: One or more matches were found.
1: No matches were found.
2: Syntax errors or inaccessible files (even if matches were found).

Note: `ed.c` contains goto statements! (Needless to say, your code should remove them, but don't get crazy here and cut them all out at once. Proceed cautiously, and remove them one by one AFTER you have made the code much simpler.)

Any unneeded variables, functions, and so on should be removed. Not doing so will cost you points.
Use a header file to prototype all functions used within `grep`. Your implementation (.c) code should be inside a single file called `grep.c`. You cannot use system functions.

Recompile frequently when doing this project. Save your work in a series of intermediate files, so you can roll back your work when everything suddenly stops working (e.g., grep00.c, grep01.c, ...). Try your `grep` program using stdin (or using `./grep < testfile.txt`) before you graduate to searching multiple files. Make sure `grep` works with its goto statements intact before beginning to remove them. Divide your code sensibly into functions, especially the regexp code, so it could be used again in a later program.

This kind of project is typical of what new developers are asked to do: make small changes to a big program. Of course, you have to understand the scope of the program you're changing before changing it, and make sure you're not breaking it. (Hint: of course you will break it, and some of the fun is in fixing it again, and trying a different approach until you get the whole thing working.) Code unrelated to regular expressions has to go.

For those unfamiliar with `ed` or `grep`, check out the manual pages for them (`man ed(1)` and `man grep(1)`).

The basic form of using `grep` is as follows: `grep search_string (options) search_files`
Here are some examples from https://alvinalexander.com/unix/edu/examples/grep.shtml

search for a string in one or more files
```
grep 'fred' /etc/passwd grep fred /etc/passwd grep null *.scala
```

regular expressions
search for lines containing 'fred' in /etc/passwd
quotes usually not when you don't use regex patterns

search multiple files
find 'fred', but only at the start of a line
find Foo or Goo in all files in the current dir

```
grep '^fred' /etc/passwd
grep '[FG]oo' *
grep '[0-9][0-9][0-9]' * #  find all lines in all files in the current dir with three numbers in a row
```

You do NOT have to implement the following functions (although they are useful)...
case-insensitive
```
grep -i joe users.txt
```

display matching filenames, not lines
```
grep -l StartInterval *.plist grep -il StartInterval *.plist
```

show matching line numbers
```
grep -n we gettysburg-address.txt
```

reverse the meaning
find joe, Joe, JOe, JOE, etc.
show all filenames containing the string 'StartInterval'

```
grep -v fred /etc/passwd   //  find any line *not* containing 'fred'
grep -vi fred /etc/passwd  //  same thing, case-insensitive
```

Submission
Turn in the code for this project by uploading all of the source files you created to a single public repository on GitHub. While you may discuss this assignment with other students, work you submit must have been completed on your own.

To complete your submission, print the sheet at the back of this file, fill out its spaces, and submit it to the instructor in class by the deadline. Failure to follow the instructions exactly will incur a 10% penalty on the grade for this assignment.

CPSC 223C Project 1 – grep (from ed), due Wednesday, 25 Mar 2020		
Your name:		
Repository *(print)*: ________________________________ `.github.io`		
Finished	**Not finished**	**Verify each of the following items with a corresponding checkmark Incorrectly marked items will incur a 5% penalty per item**
☐	☐	Researched `grep` on the Unix man page for `grep (type man grep)`
☐	☐	Researched ed on the Unix man page for ed `(type man ed for ed(1))`game.
☐	☐	**Researched regex (regular expressions), and have experimented with using them in ed. Read through in detail the source code for ed (ed.c).**
☐	☐	Duplicated ed.c's 1700+ lines of code, **saved it as grep.c,** and compiled it successfully.
☐	☐	Have used gdb or lldb debugger to step through code, and can demonstrate knowledge
☐	☐	Have compressed source code to less than 600 lines, to have multiple expressions on the same line, to aid in understanding scope and function of each part of ed.
☐	☐	Contrasted your version and Unix's ed editor, and confirmed grep runs identically.
☐	☐	Changed your version's main so the program's user interface acts like `grep`, not like ed.
☐	☐	Identified the code unlikely to be associated with the `grep` functionality, commented it out, and confirmed the code's grep features still work correctly.
☐	☐	Used a header file to prototype all functions in `grep`.h
☐	☐	Removed all unnecessary variables and functions from `grep`.c
☐	☐	Confirmed the following functionality of your version of grep: `'^Fred'` **search for Fred at beginning of line** `'Fred.$'` **search for Fred. at end of line** `'[FG]oo' *` **search for Foo or Goo** `'[0-9][0-9][0-9]'` **search for a California license plate number**
☐	☐	`grep` can search for a string in one or more files
☐	☐	`grep` prints all lines (in all search files) matching the regexp string
☐	☐	`grep` prints a leading filename and colon on each line if multiple files are searched
☐	☐	Final souce code (uncompressed) is less than 700 lines (multiple declarations of same type on same line (e.g., int *pa, *pb, c, *pd;) is ok. If statements with one expression on one line, or functions with one line are ok (e.g. int min(int a, int b) { return a < b ? a : b; })
☐	☐	Final source code (uncompressed) is less than 600 lines
☐	☐	Final source code (uncompressed) is less than 500 lines
☐	☐	Final source code (uncompressed) is less than 400 lines
☐	☐	The Project directory has been pushed to the above GitHub repository
Your comments		