CS 720/820 – Russell
Fall Semester, 2013
Assignment 1
Due: Monday, September 23, 2013

Using ANSI-C, POSIX, the getopt() interface, and a simplified version of the Boyer-Moore algorithm (to be explained in class), write a program called "plcs" — Print Lines Containing String. This program will select lines from a set of files according to a search criteria specified through the use of command-line switches and parameters, and will then write output as specified through the use of command-line switches.

The format of the command line to invoke "plcs" is as follows:

```
plcs [options] search_string [list of input file names]
```

`[options]`
: is an optional set of switches and their parameters, as detailed below, and must be processed in the order given using getopt().

`search_string`
: is a required string containing 0 or more characters.

`[list of input file names]`
: is an optional list of file names that will be read by "plcs" in the order given. Input is taken from the standard input if this list is empty. A file name consisting solely of a minus (-) means "now read data from standard input", and can appear more than once in [list of input file names].

When `[options]` is empty the action of "plcs" is to search through its list of input files, selecting each line in each input file that contains at least one exact (i.e., case sensitive) occurrence of `search_string` anywhere in that line. An input line is selected only once, regardless of the number of times `search_string` appears in it. Selected lines are written to standard output.

The following switches may appear in `[options]` to modify the selection of input lines and the format of the output lines as specified below. Switches can appear in any order and can appear any number of times. Switches are scanned in a left-to-right order and the same switch may appear multiple times.

-h Print to standard output a small amount of help information on how to use "plcs" and then exit. This output should include the meaning of the required and optional command-line components and switches.

-b An input line is selected only if `search_string` starts at the beginning of the line.

-e An input line is selected only if `search_string` ends at the end of the line.

-i The search for `search_string` is done ignoring the case of letters in both `search_string` and the input line (i.e., in a case-insensitive manner, so that, for example, This, THIS, this, and thIS are all equal).

-p Include the real path name of the input file followed by a colon and space at the start of lines written to standard output. Silently ignore this switch when reading input files from standard input.

-v Invert the choice of which lines get printed (i.e., print input lines that are NOT selected).

-l **number**
: where **number** must be positive and less than 4096. Deal with long input lines by reading them in segments, such that all initial segments (those that do not end with end-of-line) contain this number characters, and the final segment contains no more than this number characters (including any end-of-line). Then treat each segment as a separate line (i.e., supply an end-of-line at the end of each segment that doesn't already have one, and count each segment in the input line number sequence). This number does NOT include the '\0' character at the end of input lines returned by fgets(). If the "-l" switch is not given, use a default number of 255 (so that the default buffer allocated and supplied to fgets() would be 256 bytes).

-m **number**
: where **number** must be positive. Stop processing an input file (and go on to the next input file, if any) after selecting this maximum **number** of lines from that file.

-n **number**

where **number** must be positive and less than 16. When printing a line, print the sequence number of the input line within the current input file (restarting from 1 with the first line of each input file). This sequence number should follow any printing on that line caused by a -p switch, and should be immediately followed by a colon (:) and a space, and then the text of the line. **number** is the minimum column width in which to write the line number (i.e., if **number** is 5 and the input line number is 23, write 23 right-justified in a 5 character column preceding the colon; if **number** is 3 and the input line number is 654321, write 654321 in a 6 character column preceding the colon (i. e., in the minimum number of columns needed to represent the number correctly)). If both -p and -n switches are present, the output format is:

```
realpath: lineno: text
```

when the input is not from standard input, and will be:

```
lineno: text
```

when the input is from standard input.

Miscellaneous details:

— Print a meaningful error message to standard error for each illegal option and/or option value, but after detecting an illegal option or value, continue to process the remaining options (and to print error messages for those that are illegal) until all the options (but nothing else on the command line) have been processed, at which point your program should exit with failure status.
— A switch with no parameters (-b, -e, -i, -p, -v) causes the option to be set, not toggled, each time it occurs.
— A switch with parameters (-l, -m, -n) may appear multiple times. Check all occurrences for errors, but during input file processing use only the parameter value in the last occurrence.
— Errors encountered while processing `[list of input file names]` should print a meaningful error message to standard error (for example, “No such file”, or “Permission denied”, etc.), and then your program should continue processing with the next name in `[list of input file names].`
— Include the name of an offending file, switch, and/or switch parameter in all error messages regarding that file, switch, and/or switch parameter.
— The “-b” and “-e” switches are not exclusive — if they both appear in the same command line, it means that `search_string` should match lines that contain only that `search_string` and nothing else.
— The end-of-line character is ’\n’. It does not participate in matching against `search_string.`
— Reading from standard input terminates when end-of-file is detected, just as for any named file.
— Ensure that multiple occurrences of "-" as a name in the command line actually cause the standard input to be read multiple times, rather than having the second, third, etc. read immediately detect the end-of-file left over from a previous read from standard input. This should also work with input redirection.
— Lines written to the output can become very long. Do not worry about this, just let them “wrap” over extra screen lines if they are written to the screen.

You must create a shell script called “dotest” that contains a sequence of tests to demonstrate the error detection facilities of your program as well as its correct operation. This shell script should produce output that says what each test tests, and that states clearly and unambiguously when the test succeeds or fails (for example, a test that gives an erroneous switch to “plcs” will succeed only if “plcs” fails).

You must also create a Makefile that can be used by gmake to build your executable on all platforms. The executable must be called “plcs”.

Using the gcc compiler, your program should compile and run with no changes on the three different architectures and operating systems represented by agate.cs.unh.edu, newton.cs.unh.edu, and topaz.cs.unh.edu.

Submit all your source files (no object or executable files, please), your Makefile, your “dotest” script and any test input files by typing:

```
~cs720/submit 1 file1 file2 file3 ...
```

where 1 is the number of this assignment. You MUST be on agate.cs.unh.edu to run submit! Please put your name in every file you submit.