# CSCI415—Systems Programming

Spring 2017

Programming Assignment 2
RGPP, a Recursive Grep Post-Processor

## Problem Statement

Unix utilities tend to produce output which has been described as "compact" by fans of Unix and as "user-unfriendly" by its detractors. This assignment will have you make the output of a common systems utility more palatable to a user and to add value to the utility itself.

`grep` is a most useful utility which scans specified files for lines which contain a specified pattern. Do a "man grep" to see the on-line manual page. You are to develop a program called `rgpp` (for **r**ecursive **g**rep **p**ost-**p**rocessor) which behaves roughly as follows:

- It takes the output of a run of `grep -r -H -n -s -I -i pattern directory` on its standard input. We assume that the `pattern` is just a string (not a regular expression). We can describe the semantics of this command argument by argument.
  - `-r`: `grep` will be run recursively against all files in the subtree of the filesystem rooted at the designated `directory`;
  - `-H`: file names will be output for each matching line;
  - `-n`: line numbers where matches occur will be produced;
  - `-s`: error messages about inaccessible or unreadable files will not be produced;
  - `-I`: binary files are not searched (treat binary files as if there were no matches in them);
  - `-i`: the pattern will be matched in case-insensitive mode;
- `rgpp` will produce, on its standard output:
  - A "banner line" with a suitable message and a count of the number of lines in the `grep`-searched files which matched the specified pattern, followed by a listing of each file in which a match occurred with each line optionally numbered and all lines matching the `grep` search string clearly tagged (perhaps preceded by a "−− >" or similar string).
  - A header line should be output before the listing of a file in which there is a match. That header line should include the name of the file. An example of an appropriate header line can be seen in the examples linked from the assignments web page.

Having given you a rough idea of the operation of `rgpp`, the following is a statement of its functionality relative to command line arguments which control `rgpp`'s operation. The `rgpp` program is invoked as:

```
rgpp [-l | -w word] {-n} {-b}
```

The first command argument must be a `-l` or a `-w`. A `-l` means that `rgpp` is operating in "line mode," behaving much as described above. Lines from files which correspond to line hits from `grep` will be tagged. The `-n` is optional: if present, the file listing will be line-numbered; if absent, the lines will be output without a leading line number. The `-b` is optional: if present, the banner line stating the number of lines hit by `grep` will be produced *before* the listing of the files. If there is no `-b` then there is no banner line.

Note that the optional `-b` and `-n` may appear together, and in either order. Your program should not impose a required order.

If `rgpp` is invoked with `-w` as the first command line argument, it operates in "word mode." In word mode, the banner, if requested, states the number of *non-overlapping* instances of the designated pattern in the files before emitting the numbered, tagged, listings. On tagged lines (lines where a match to the pattern must be found), the non-overlapping instance(s) of the pattern must be highlighted. Under the assumption that `rgpp` will be run within an xterm, the string `\e[7m` sent to the standard output will switch it into inverse video mode. The string `\e[0m` switches the xterm back to normal (non-inverted) video mode. A pattern is "highlighted" by rendering it in inverse video. The output is optionally line-numbered as above, depending on the presence of the `-n` switch.

Bear in mind that this code is to be written as a system utility, and, as such, should "handle" all possible errors (e.g., it is possible that what is believed to be `grep` output contains line numbers which are non-numeric or beyond the highest numbered line in the program, an unreadable file containing `grep`'s output may be specified by an idiot user, etc.). You are responsible for producing code which is robust in the presence of ignorant, stupid, or even malicious, use by the user community. Your program will be tested against bizarre input when it is graded. Please note that there will be some stupid uses of **`rgpp`** with which you cannot reasonably deal. For example,

```
 % grep -r -H -s -I -i xxx . | rgpp -l -b
```

violates the restriction that `grep` must be invoked with the command line arguments as specified above (`-n` is missing in this case);

```
 % grep -r -H -n -s -I -i xxx  . > grepout; cd somewhere; rgpp -l < grepout
```

is even more difficult to handle (given the way that `grep` embeds file names in its output), and

```
 % grep -r -H -n -s -I -i xxx . | gpp -w yyy -b
```

is mass confusion. Do the best that you can.

The operative rule is that `rgpp` should not dump core, but it is not expected to produce good output when invoked incorrectly. Finally, generate informative error messages when doing so doesn't require an inordinate amount of work.

Your implementation of `rgpp` should not impose limits on the number or size of the files or the number of lines on the standard input. In operational terms, this means that you must dynamically allocate some of the major data structures required for the implementation of `rgpp`.

Check the web page for CSCI415 assignments for examples of the operation of `rgpp`.

## Due

11:59pm on Monday, February 13. Be sure to keep your program in a single source file named `rgpp.c`. Details on how to submit your program will be posted near the due date.