Lab-5: Text-based programming, http://en.wikipedia.org/wiki/AWK

Mar.-19, 2010

> *"AWK is a language for processing files of text. A file is treated as a sequence of records, and by default each line is a record. Each line is broken up into a sequence of fields, so we can think of the first word in a line as the first field, the second word as the second field, and so on. An AWK program is of a sequence of pattern-action statements. AWK reads the input a line at a time. A line is scanned for each pattern in the program, and for each pattern that matches, the associated action is executed."* – Alfred V. Aho

The design of nawk was influenced by shell scripting languages that we already used. In turn nawk influenced the development of more advanced and complex languages such as perl. It shares much syntax with C, but nawk is not a compiled language to be used on large data files. It is, however, extremely powerful on short or text-based files for advanced searches or data extraction. It can be executed from a command-line or from within a shell script as

```
% nawk ‘pattern {action}’ infile >outfile
```

The “pattern” is basically a statement, which, if true, will enable the “action.” A set of one-line programmes, variations of which I use a lot, are:

```
% nawk ‘END{print NR}’ infile             #NR is a line number, printed after all of infile is read
% nawk ‘$0 ~ “string” {print $0}’ infile  # if a line matches “string”, print that line
% nawk ‘{print NR, NF}’ infile            # for each line, print line number and number of column
% nawk ‘$5 == 4 {print sqrt($5)}’ infile  # if column 5 equals 4, print the square root of 4
```

The program content ‘pattern {action}’ can also placed within file which can executed either from a command line or from within a shell script as

```
% nawk –f  file.awk par=set_pa infile >outfile
```

where the passing of a variable called par with a value set_par is optional as is the direction of the output to outfile. Check the Wiki page above (and/or google awk) for more details.

You are here tasked to write an awk-file called RadianScales.awk (ReflecScales.awk) to extract the scaling factors that convert the MODIS science integer values to radiance (reflectance) values. For this purpose you will make extensive use of the file HeaderInfo.ascii:

14.	Extract relevant information on calibration, file size, scan numbers, etc, from a header dump, e.g. `hdfdump –h $name.L1B_LAC >HeaderInfo.ascii`
15.	How many lines does the HeaderInfo.ascii file contain? Are the number of column per line constant?
16.	Search the HeaderInfo.ascii file from the command line for any occurance of a text string “radiance”: `nawk ‘$0 ~ “string” {print $0}’ HeaderInfo.ascii`
17.	You will get too many lines, so your string “pattern” was too general, modify it to, say, “radiance_scales”
18.	The output may still be too cryptic, so search also for the text string “band_names” and see if you can make sense of the different radiance scales. How many bands are there? What’s special about band-26?
19.	Extract the radiance_scales for the bands 1 and 2 by designing a unique “pattern” that returns only one line.
20.	Note that the scales contain the letter “f” at the end, print a sub-string without it; the “action” `print substr($x,StartCharacter,NumberOfCharacters)` will be useful, here “x”, “StartCharacter”, “NumberOfCharacters” are integers you must specify.
21.	Make file RadianScales.awk that contains the above and add the following line to your proc2.csh shell script: `nawk –f RadianScales.awk HeaderInfo.ascii >log.dat` where you replace the ‘pattern {action}’ construct with { if (pattern);{action};endif }