awk is a standard Unix utility which has its own script language to provide data extraction, transformations, and reports. Its name comes from the last names of its creators (Aho, Weinberger, Kernighan). The GNU version of awk is named **gawk**.

Text files are treated as lines of data records, each having possibly many fields. Whitespace (i.e., spaces and/or tabs) is the default delimiter between fields. The delimiter can be changed by setting **FS**, which is the field separator.

The actions to be performed can be in either the command line directly or can be a program script inside a *programFile*:

```
awk options 'program' file1 ...
awk options -f programFile file1 ...
awk options -f programFile -
```

The third form for invoking awk tells awk that the input comes from stdin.

An awk program is a series of pattern action pairs:

condition1 { *action1* }
condition2 { *action2* }
...

If the *condition* is true, the corresponding *action* is executed. If the *condition* is omitted, the *action* is unconditionally executed. Each *action* can be a series of commands.

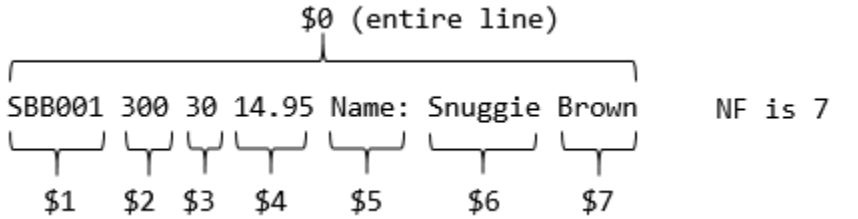
The most common conditions are pattern matches similar to sed patterns; however, awk also makes it easy to apply a pattern to a particular field:

$k ~ *pattern*
$*k* !~ *pattern*

Comments in awk begin with a "#".

Create an `AwkExamples` directory. When logged into a fox server, please cd to the **`/usr/local/courses/ssilvestro/cs3423/awk`** directory and copy all the files to your awk directory.

Example 1: We want the product ID (first field) and unit price (fourth field) from the inventory.txt file. Sample data line:




```
$ gawk '{print $1, $4}' inventory.txt
PPF001 9.95
SBB001 14.95
SBG002 14.95
BOM001 29.95
MCW001 12.45
TTP001 9.95
NHC001 9.95
SSX001 29.95
```

Note that example #1 doesn't have a condition; therefore, the action applies to every line. Also, notice that we can easily change the order of the columns in the **print**.

Example 2: Show the product ID and unit price for any products containing "Snuggie".

```
$ gawk '$0 ~ /Snuggie/ {print $1, $4}' inventory.txt
   --- functionally equivalent ---
$ gawk '/Snuggie/ {print $1, $4}' inventory.txt

SBB001 14.95
SBG002 14.95
```

Example 3: Show the product ID and unit price for any products having a unit price that ends in ".95".

```
$ gawk '$4 ~ /\.95$/ {print $1, $4}' inventory.txt
PPF001 9.95
SBB001 14.95
SBG002 14.95
BOM001 29.95
TTP001 9.95
NHC001 9.95
SSX001 29.95
```

Using a Program File

As shown above, the **-f** switch is used to specify a program file which allows for more complex capabilities. awk provides C-like **if**, counting **for**, **while**, and **printf** action commands. It also supports a **for in** to iterate over the contents of an array.

Some special conditions:

BEGIN executes the action before the records are read. In the actions, we typically initialize variables and print column headings.

END executes the action after the records are read. It is very common to print totals in the action corresponding to an END condition.

Example 4: Show the product ID, inventory quantity, and unit price for any products having an inventory quantity (second field) greater than 200. Also show a column heading.

```
$ cat >example4
BEGIN {printf("%-6s %4s  %-10s\n", "ID", "QTY", "UNIT PRICE");}
$2 > 200 {printf("%6s %4d  %8.2f\n", $1, $2, $4);}
CTRL-D
$ gawk -f example4 inventory.txt
ID      QTY  UNIT PRICE
SBB001  300     14.95
SBG002  400     14.95
NHC001  300      9.95
```

Example 5: Show the product ID and the product description for every product. Since a product description can be many words, we will use a counting for loop beginning at $6 and ending at NF. Note: you can reference $i.

```
$ cat >example5
{
    # output the product ID
    ??
    # output each word in the product description
    ??
    printf("\n");
}
CTRL-D
$ gawk -f example5 inventory.txt
PPF001 Popeil Pocket Fisherman
SBB001 Snuggie Brown
SBG002 Snuggie Green
BOM001 Bass-O-Matic
MCW001 Miracle Car Wax
TTP001 Topsy Turvy Planter
NHC001 Electric Nose Hair Clipper
SSX001 Secret Seal
```

The awk arithmetic operators are from the C programming language.

The type of comparison (numeric or string) is based on the operands. If both are numeric, a numeric comparison is done. Otherwise, a string comparison is used.

Example 6: For products having a unit price greater than $10 and more than 100 items in inventory, print the ID, inventory quantity, unit price, and gross value (product of inventory quantity and unit price). Also print the total gross value for all products meeting the criteria.

```
$ cat >example6
BEGIN {
          printf("%-6s %4s  %-10s  %-12s\n", "ID", "QTY", "UNIT PRICE",
               "GROSS PRICE");
          total = 0;
      }
{
      if ( $2 > 100 && $4 > 10.0 )
      {
          gross = $2 * $4;
          printf("%6s %4d  %8.2f    %10.2f\n", $1, $2, $4, gross);
          total = total + gross;
      }
}
END { printf("%-6s %4s  %-10s  %10.2f\n", " ", " ", " ", total);}
CTRL-D
```

```
$ gawk -f example6 inventory.txt
ID      QTY  UNIT PRICE  GROSS PRICE
SBB001  300     14.95       4485.00
SBG002  400     14.95       5980.00
SSX001  150     29.95       4492.50
                           14957.50
```

How can we rewrite this to not include the if-statement? ??

Exercise#1: List the login names for anyone in the faculty group (group 1000). Use the following to get all users:

```
$ getent passwd
```

It returns records that look like this:

```
krobbins:x:512:1000:Kay A. Robbins:/home/krobbins:/bin/csh
maynard:x:511:1000:Hugh B. Maynard:/home/maynard:/bin/tcsh
clark:x:1000:1000:Larry Clark:/home/clark:/bin/tcsh
abc123:x:5035:1001:Bob:/home/abc123:/bin/tcsh
...
```

The faculty group is group 1000 which is in the 4th field.

How will your awk program get its input? ??
How do we specify a different field separator? ??

```
$ ??
??
??
```

Range Patterns

awk supports ranges of lines as a pattern just like sed:

```
/pat1/,/pat2/
```

Example 7: Analyze the code in cs1713p0.c, counting number of comment lines, number of blank lines, and number of code lines.

```
$ cat > example7
BEGIN { blankCount = 0; commentCount=0}
    /^\/\*/,/\*\// { commentCount++ }
    /^[ \t]*\/\// { commentCount++}
    /^[ \t]*$/ {blankCount++}
END {print "Total Lines:", NR;
     print "Comment lines:", commentCount;
     print "Blank lines:", blankCount;
     print "Code: ", NR - commentCount - blankCount;
    }
CTRL-D
$ gawk -f example7 cs1713p0.c
Total Lines: 77
Comment lines: 26
Blank lines: 9
Code:  42
```

Associative Arrays

Awk supports associative arrays (i.e., hash tables). The key for an associative array can be a character string. To assign a value:

```
array[key] = value;
```

To check whether an entry exists:

Example 8: print the total of purchase items for each item requested. Examine invCommand.txt. We are only interested in the ORDER ITEM records.

```
$ cat >example8
$1 == "ORDER" && $2 == "ITEM"{
        if ($3 in invM)
            invM[$3] += $4;
        else
            invM[$3] = $4;
```

```
if (key in array)
    doSomething;
```

To iterate over the keys of the array:

```
for (key in array)
    doSomething;
```

```
}
END {
        for (key in invM)
            print key, invM[key];
    }
CTRL-D
$ gawk -f example8 invCommand.txt
XXX001 20
SBG002 410
SVC001 3
APC001 1
NHC001 260
BOM001 2
PPF001 11
MCW001 62
SBB001 38
```

Example 9: produce a shell script

In this example, the output from awk will be a shell script.

It is very common to have core dump files named "core" taking up lots of space throughout your directories. Can we use awk to help remove the files?

As a system administrator, you would use "locate core" to find the files which would give us a huge result containing many files (not just mine).

To simplify, we will simulate the use of locate by using find.

First lets find core files using find:

```
$ find ~ -name '*core*'
/home/ssilvestro/cs1713/Pgm3/core
/home/ssilvestro/cs4713/core
/home/ssilvestro/cs2123/core
/home/ssilvestro/.cache/compizconfig/core.pb
/home/ssilvestro/core
/home/ssilvestro/cs3423/score
/home/ssilvestro/cs3423/core
```

Notice that some of the files are `core`, but others just have "`core`" somewhere in the name. We don't want to remove the other files. We will also confirm that each file starts with `/home/ssilvestro/` since locate would have returned others.

Example 9: produce a shell script to remove files

What should our awk program do?

```
$ cat > example9
BEGIN { count = 0;}
/^\/home\/ssilvestro\// && /\/core$/ {

   ??

}
END ??
```

CTRL-D

```
$ find ~ -name '*core*' | awk -f example9 - > coreRm

$ bash coreRm
removed 5 files
```

Suppose we have awk output a shell script file that contains

```
rm /home/ssilvestro/cs1713/Pgm3/core
rm /home/ssilvestro/cs4713/core
rm /home/ssilvestro/cs2123/core
rm /home/ssilvestro/core
rm /home/ssilvestro/cs3423/core
echo "removed 5 files"
```

We can then run that script file after verifying it.

Passing arguments into awk code

We can pass variable values into awk by specifying:

`awk options 'program' -v 'var=value' file1`

We can change example9 to allow us to pass in the user.

The awk function match returns true if the functions the first argument matches the pattern specified in the second argument.

Example 10: passing in a variable for the /home/*user*/

```
$ cat > example10
BEGIN { count = 0;}
/\/core$/{
    if ( match( $0, arg1 ) )
    {
        print "rm", $0;
        count++;
    }
}
END {print "echo removed", count, "files"}
```

CTRL-D

```
$ find ~ -name '*core*' | awk -f example10 -v 'arg1=/home/ssilvestro/' -
```

Why use `match()`?

??

Special Variables

FS	input field separator (defaulted to white space)
OFS	output field separator (defaulted to blank)
RS	input record separator (defaulted to \n)
ORS	output record separator (defaulted to \n)
NF	number of fields for the current line
NR	record number of the current line

Some built-in functions

`int(val)`	returns the truncated integer value
`length(val)`	returns the length of the value
`index(str,match)`	returns the index of *match* in *str* or 0 if it isn't found
`substr(str,pos,length)`	returns the substring of *str* beginning at *pos* for *length* characters
`split(string, array [, fieldsep])`	splits string into array (indexed at 1) according to fieldsep
`sub, gsub, gensub`	see gawk manual for full details