CS29206 Systems Programming Laboratory
Spring 2024

Introduction to gawk

Abhijit Das
Pralay Mitra

The Unix command awk

- Named after the designers Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan
- We discuss the GNU version gawk
- awk is a full-fledged programming language
- Before the advent of Perl, awk used to be the most powerful pattern processing language
- Run the command as

```
gawk <OPTIONS> 'COMMANDS' <FILE(S)>
```

- If the commands are written in `COMMANDFILE`, run as

```
gawk <OPTIONS> -f COMMANDFILE <FILE(S)>
```

Records and fields

- awk reads the input file(s) line by line.
- Each line is called a record.
- Each record is split into fields.
- The default field separator is space or tab.
- You can specify your separator by running with the –F option.

```
gawk -F: 'COMMANDS' <FILE(S)>
```

- The current record is accessed as `$0`.
- The individual fields are accessed as `$1`, `$2`, `$3`, . . .

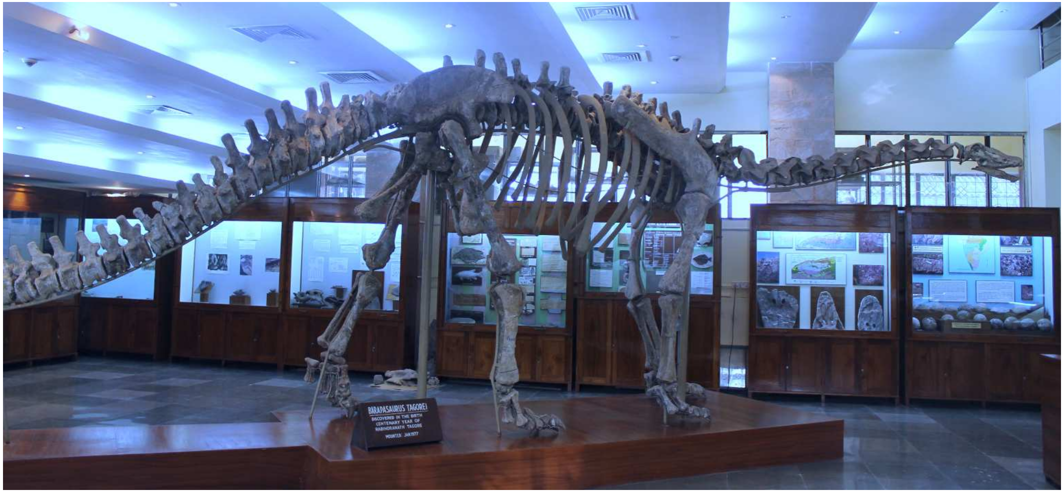
An example file

```
Alamosaurus:sauropod:21::H:70-65:USA
Albertaceratops:ceratopsian:7::H:80-75:Canada, USA
Albertosaurus:large theropod:9:1500:C:76-74:Canada
Allosaurus:large theropod:12:2000:C:156-144:Portugal, USA
Ankylosaurus:armored dinosaur:7:4000:H:74-67:Canada, USA
Antarctosaurus:sauropod:18::H:84-65:Argentina, Chile, Uruguay
Apatosaurus:sauropod:21::H:154-145:USA
Aragosaurus:sauropod:18::H:132-121:Spain
Archaeopteryx:small theropod:0.5::C:147:Germany
Argentinosaurus:sauropod:35:70000:H:90:Argentina
Avaceratops:ceratopsian:2.3::H:80-75:USA
Bactrosaurus:euornithopod:6::H:84-71:China
Barapasaurus:sauropod:14::H:185-170:India
Barosaurus:sauropod:24::H:155-145:Tanzania, USA
Baryonyx:large theropod:10:2000:C:125:Spain, UK
Brachiosaurus:sauropod:30::H:155-140:Algeria, Portugal, Tanzania, USA
Carcharodontosaurus:large theropod:15::C:98-94:North Africa
Carnotaurus:large theropod:7.6::C:70:Argentina
Centrosaurus:ceratopsian:6:1000:H:76-74:Canada
Ceratosaurus:large theropod:6:970:C:153-148:Portugal, USA
Chindesaurus:small theropod:4::C:227-210:USA
Coelophysis:small theropod:2:27:C:225-190:South Africa, USA, Zimbabwe
Dacentrurus:armored dinosaur:6::H:154-150:France, Portugal, UK
Deinocheirus:large theropod:10::O:70-66:Mongolia
Deinonychus:small theropod:3:75:C:120-110:USA
Dilophosaurus:large theropod:6:300:C:190:USA
Diplodocus:sauropod:26:15000:H:155-145:USA
Edmontosaurus:euornithopod:13:3400:H:76-65:Canada
Gastonia:armored dinosaur:4.6::H:142-127:USA
Giganotosaurus:large theropod:12.5:8000:C:112-90:Argentina
```

An example file (continued)

```
Gobisaurus:armored dinosaur:5::H:121-99:China
Hadrosaurus:euornithopod:9::H:78-74:USA
Heterodontosaurus:euornithopod:1.2::H:205:Lesotho, South Africa
Iguanodon:euornithopod:10:4000:H:140-110:Belgium, UK
Indosuchus:large theropod:7::C:71-65:India
Isisaurus:sauropod:::H:71-65:India
Kentrosaurus:armored dinosaur:5::H:155-150:Tanzania
Kotasaurus:sauropod:9::H:205-190:India
Leptoceratops:ceratopsian:3::H:67-65:Canada, USA
Majungasaurus:large theropod:6::C:84-71:Madagascar
Megalosaurus:large theropod:9::C:170-155:UK
Microraptor:small theropod:0.8:1:C:125-122:China
Monolophosaurus:large theropod:5.7::C:180-159:China
Oviraptor:small theropod:2:20:O:85-75:Mongolia
Parasaurolophus:euornithopod:11:3500:H:76-74:Canada, USA
Patagosaurus:sauropod:18::H:164-159:Argentina
Pentaceratops:ceratopsian:6.8::H:76-74:USA
Plateosaurus:sauropod:7:4000:H:210:France, Germany, Switzerland
Protoceratops:ceratopsian:1.8:400:H:74-70:China, Mongolia
Riojasaurus:sauropod:5.15::O:221-210:Argentina
Scutellosaurus:armored dinosour:1.2::H:205-202:USA
Sinraptor:large theropod:7.6::C:169-142:China
Spinosaurus:large theropod:18:4000:C:95-70:Egypt, Morocco
Stegosaurus:armored dinosaur:9::H:155-145:USA
Tarbosaurus:large theropod:10::C:74-70:China, Mongolia
Thecodontosaurus:sauropod:2.5::O:227-205:UK
Triceratops:ceratopsian:9:5500:H:68-66:USA
Tyrannosaurus:large theropod:12:7000:C:68-66:Canada, USA
Utahraptor:large theropod:6:1000:C:112-100:USA
Velociraptor:small theropod:1.8:7:C:74-70:Mongolia
```

Barapasaurus Tagorei



An example record

- Consider the line

```
Indosuchus:large theropod:7::C:71-65:India
```

- Here : is used as the field separator.
- We have the strings stored in the following variables.

```
$0  =  "Indosuchus:large theropod:7::C:71-65:India"
$1  =  "Indosuchus"
$2  =  "large theropod"
$3  =  "7"
$4  =  ""
$5  =  "C"
$6  =  "71-65"
$7  =  "India"
```

$0 gives the whole line

and after that we can get each fields by the $number
Note that the $4 is empty as we had FS=":"

The commands

- There is an optional BEGIN section that is executed before any record is read.
- This is followed by reading the records one by one, and performing actions driven by a set of patterns.
- Finally, there is an optional END section that is executed after all records are read.

An awk program

```
BEGIN { Initial actions }
PATTERN1 { Action1 }
PATTERN2 { Action2 }
...
PATTERNn { Actionn }
END { Final actions }
```

- For each record, only those actions are taken for which the record matches the corresponding patterns.
- The actions are taken in the sequence given in the program.
- An empty pattern matches every record.

A simple awk program

details.awk

```
BEGIN {
   FS = ":"
   print "Going to read the dinosaur database..."
}
{ print $1 }
{ print "\tType: " $2 }
{ print "\tLength: " $3 " meters" }
$4 == "" { print "\tWeight: Unknown" }
$4 != "" { print "\tWeight: " $4 " kilograms" }
$5 == "H" { print "\tDiet: Herbivorous" }
$5 == "C" { print "\tDiet: Carnivorous" }
$5 == "O" { print "\tDiet: Omnivorous" }
{ print "\tLived " $6 " million years ago" }
{
   print "\tFossils found in" }
   n = split($7, clist, ", ")
   for (i=1; i<=n; ++i) { print "\t\t" clist[i] }
}
END { print "That is all I have. Bye..." }
```

The pattern commands gets executed for each line multiple times

Indexing is 1-based

Running this awk script

```
$ gawk -f details.awk dinosaurs.txt
Going to read the dinosaur database...
Alamosaurus
        Type: sauropod
        Length: 21 meters
        Weight: Unknown
        Diet: Herbivorous
        Lived 70-65 million years ago
        Fossils found in
                USA
Albertaceratops
        Type: ceratopsian
        Length: 7 meters
        Weight: Unknown
        Diet: Herbivorous
        Lived 80-75 million years ago
        Fossils found in
                Canada
                USA
...
Velociraptor
        Type: small theropod
        Length: 1.8 meters
        Weight: 7 kilograms
        Diet: Carnivorous
        Lived 74-70 million years ago
        Fossils found in
                Mongolia
That is all I have. Bye...
$
```

Doing all the record-processing actions as a single action

```
BEGIN {
   FS = ":"
   print "Going to read the dinosaur database..."
}
{
   print $1
   print "\tType: " $2
   print "\tLength: " $3 " meters"
   if ($4 == "") { print "\tWeight: Unknown" }
   if ($4 != "") { print "\tWeight: " $4 " kilograms" }
   if ($5 == "H") { print "\tDiet: Herbivorous" }
   if ($5 == "C") { print "\tDiet: Carnivorous" }
   if ($5 == "O") { print "\tDiet: Omnivorous" }
   print "\tLived " $6 " million years ago"
   print "\tFossils found in"
   n = split($7, clist, ", ")
   for (i=1; i<=n; ++i) { print "\t\t" clist[i] }
}
END { print "That is all I have. Bye..." }
```

Filtering by pattern matching

- The pattern can be any regular expression.
- Enclose the pattern by a pair of delimiters (usually /).

select.awk

```
BEGIN {
   FS = ":"
   nIndian = 0
   nlarge = 0
   nsmall = 0
}
{
   if ($7 ~ /India/) { nIndian++; Indian[nIndian] = $1 }
   if ($2 ~ /theropod/) {
      if ($2 ~ /large/) { nlarge++; LT[nlarge] = $1 }
      else { nsmall++; ST[nsmall] = $1 }
   }
}
END {
   print nIndian " dinosaurs found in India"
   for (i=1; i<=nIndian; i++) print "\t" Indian[i]
   print nlarge " large theropods:"
   for (i=1; i<=nlarge; i++) print "\t" LT[i]
   print nsmall " small theropods:"
   for (i=1; i<=nsmall; i++) print "\t" ST[i]
}
```

Output of the selection program

```
$ gawk -f select.awk dinosaurs.txt
4 dinosaurs found in India
        Barapasaurus
        Indosuchus
        Isisaurus
        Kotasaurus
18 large theropods:
        Albertosaurus
        Allosaurus
        Baryonyx
        Carcharodontosaurus
        Carnotaurus
        Ceratosaurus
        Deinocheirus
        Dilophosaurus
        Giganotosaurus
        Indosuchus
        Majungasaurus
...
        Utahraptor
7 small theropods:
        Archaeopteryx
        Chindesaurus
        Coelophysis
        Deinonychus
        Microraptor
        Oviraptor
        Velociraptor
$
```

Similarities with C

- awk syntax is quite similar to C syntax.
- Comparison operators: `==`, `!=`, `<`, `<=`, `>`, `>=`.
- **New operator:** `~` (pattern matching) and `!~` (pattern non-matching).
- Logical operators: `&&`, `||`, and `!`.
- Arithmetic operators: `+`, `-`, `*`, `/`, `%`, `++`, `--`.
- **New operator:** `**` (exponentiation).
- Assignment operators: `=`, `+=`, `-=`, `*=`, `/=`, and `%=`.
- if and if else statements.
- while and for loops, break, and continue.
- printf and sprintf work exactly as in C.

Built-in variables

$0	The current record
$1,$2,$3,. . .	The fields in the current record
RS	The record separator (default: new line)
NR	The number of the current record (1, 2, 3, . . .)
FS	Field separator
NF	The number of fields in the current record
FILENAME	The name of the current file (NULL if the input is taken from stdin)
OFS	Output field separator (default: space)
ORS	Output record separator (default: new line)

Note: The print action without any argument prints the current record. OFS and ORS are used for this printing.

Variables and arrays

- Variables are not needed to be declared before use.
- Numeric variables are automatically initialized to 0.
- String variables are automatically initialized to the empty string.
- Variables do not have fixed types.
- Strings and numbers are treated in a unified manner.
- If a string is used in a numerical context, it is automatically converted to a number if it is a numeric string, or to 0 otherwise.
- A number is automatically converted to a numeric string (like during printing).
- Strings are compared with respect to the lexicographic ordering. For example, `9 < 10` (as numbers), whereas `"9" > "10"` (even though both are numeric). lexicographical comparison is used
- Array indexing is 1-based.

Some new built-in functions

int(x)	The integer part of x
length(s)	Length of the string s
index(s,t)	Index of the substring t in the string s (0 is t is not a substring of s)
substr(s,b,l)	Substring of the string s beginning at index b and of length l
toupper(s)	Copy of the string s converted to upper case
tolower(s)	Copy of the string s converted to lower case
split(s,A,d)	Split the string s with respect to the delimiter (a string again), and store the parts in the array A. The number of parts obtained by splitting s (the size of A) is returned.

Example: Average length of sauropod dinosaurs in different periods

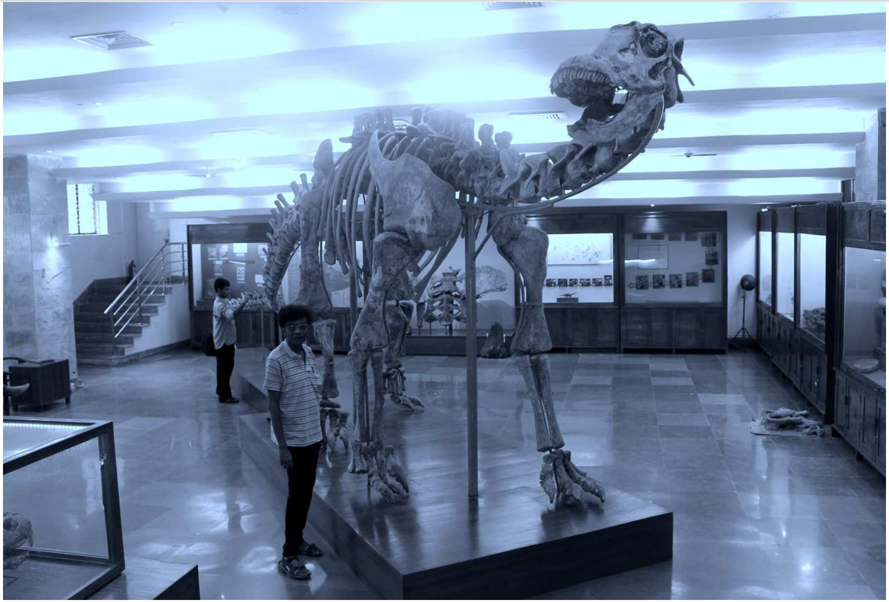
average.awk

```
{
   if ($2 == "sauropod") {
      pos = index($6,"-")
      if (pos == 0) {
         ts = te = int($6)
      } else {
         ts = int(substr($6, 1, pos-1))
         te = int(substr($6, pos+1, length($6)-pos))
      }
      if ((ts <= 252) && (te >= 201)) { nt++; sumt += $3 }
      else if ((ts <= 201) && (te >= 145)) { nj++; sumj += $3 }
      else if ((ts <= 145) && (te >= 65)) { nc++; sumc += $3 }
      else { printf("Period cannot be determined for %s (%d,%d)\n", $1, ts, te) }
   }
}
END {
   printf("Average lengths of sauropod dinosaurs\n")
   printf("   Triassic period (252-201 Ma)   : %6.2f meters\n", sumt / nt)
   printf("   Jurassic period (201-145 Ma)   : %6.2f meters\n", sumj / nj)
   printf("   Cretaceous period (145-65 Ma) : %6.2f meters\n", sumc / nc)
}
```

```
$ gawk -F: -f average.awk dinosaurs.txt
Period cannot be determined for Brachiosaurus (155,140)
Period cannot be determined for Kotasaurus (205,190)
Average lengths of sauropod dinosaurs
   Triassic period (252-201 Ma)   :   4.88 meters
   Jurassic period (201-145 Ma)   :  20.60 meters
   Cretaceous period (145-65 Ma) :  18.40 meters
$
```

The colon after -F defines the Field separator

Barapasaurus Tagorei



Associative arrays (or hashes)

- Arrays can be indexed by strings.
- The syntax is the same: Array[string]
- Here, string is not automatically converted to an integer index.
- **Note:** `Array[5]` and `Array["5"]` are different.
- Loops can be used on associative arrays as:

```
for (name in Array) {
    # Access entries as Array[name]
}
```

- Iterations are not in the sorted order of names.

Example: Country-wise listing of large theropod dinosaurs

Executable gawk script theropod.awk

```
#!/usr/bin/gawk -f
{
   if ($2 == "large theropod") {
      n = split($7, country, ", ");
      for (i=1; i<=n; ++i) { tlist[country[i]] = tlist[country[i]] " " $1 }
   }
}
END {
   for (c in tlist) {
      printf("%-15s: %s\n", c, tlist[c])
   }
}
```

```
$ ./theropod.awk -F: dinosaurs.txt
USA            :  Allosaurus Ceratosaurus Dilophosaurus Tyrannosaurus Utahraptor
Morocco        :  Spinosaurus
Egypt          :  Spinosaurus
Mongolia       :  Deinocheirus Tarbosaurus
China          :  Monolophosaurus Sinraptor Tarbosaurus
UK             :  Baryonyx Megalosaurus
Spain          :  Baryonyx
Canada         :  Albertosaurus Tyrannosaurus
India          :  Indosuchus
North Africa   :  Carcharodontosaurus
Madagascar     :  Majungasaurus
Portugal       :  Allosaurus Ceratosaurus
Argentina      :  Carnotaurus Giganotosaurus
$
```

The environment variables

- The environment variables are available in the built-in associative array ENVIRON.

environ.awk

```
BEGIN {
   for (name in ENVIRON) { printf("%s = %s\n", name, ENVIRON[name]) }
}
```

The output

```
$ gawk -f environ.awk
IM_CONFIG_PHASE = 1
DBUS_SESSION_BUS_ADDRESS = unix:path=/run/user/1000/bus
SHLVL = 2
GNOME_DESKTOP_SESSION_ID = this-is-deprecated
PWD = /home/foobar/spl/prog/awk
...
USER = foobar
DISPLAY = :0
AWKPATH = .:/usr/share/awk
...
$
```

User-defined functions and run-time user inputs

Fibonacci.awk

```
#!/usr/bin/gawk -f

function F ( n )
{
   if (n <= 1) { return n }
   return F(n-1) + F(n-2)
}

BEGIN {
   printf("Enter a positive integer: ")
   getline n < "-"
   n = int(n)
   print "Fib(" n ") = " F(n)
}
```

Running the program

```
$ ./Fibonacci.awk
Enter a positive integer: 10
Fib(10) = 55
$ ./Fibonacci.awk
Enter a positive integer: 20
Fib(20) = 6765
$
```

Scope of variables

- All variables used are global. the most important point to be noted while coding
- There is no provision for declaring local variables.
- Only the function parameters act as local variables.
- Parameter passing is by value only.
- If you want to use local variables in a function, do the following.
  - Add your local variables to the list of parameters.
  - You do not need to pass values to all the parameters.
  - Any value not passed is initialized to 0 or the empty string.

All variables are global

nolocal.awk

```
#!/usr/bin/gawk -f

function oddsum ( n )
{
   print "oddsum(" n ") called"
   sum = 0
   term = 1
   for (i=1; i<=n; ++i) {
      sum += term
      term += 2
   }
   return sum
}

BEGIN {
   n = 10
   sum = 0
   for (i=1; i<=n; ++i) { sum += oddsum(i) }
   print sum
}
```

- The output is 162.
- `n` is a local variable, but `i` and `sum` are global variables in `oddsum()`.

What happens to *i* and *sum*

```
#!/usr/bin/gawk -f

function oddsum ( n )
{
   sum = 0
   term = 1
   for (i=1; i<=n; ++i) {
      sum += term
      term += 2
   }
   return sum
}

BEGIN {
   n = 10
   sum = 0
   for (i=1; i<=n; ++i) {
      print "Calling oddsum(" i ")"
      sum += oddsum(i)
      print "sum = " sum
   }
   print sum
}
```

Output

```
Calling oddsum(1)
sum = 2
Calling oddsum(3)
sum = 18
Calling oddsum(5)
sum = 50
Calling oddsum(7)
sum = 98
Calling oddsum(9)
sum = 162
```

Fixing the problem

```
#!/usr/bin/gawk -f

function oddsum ( n, i, sum )
{
   print "oddsum(" n ") called"
   sum = 0
   term = 1
   for (i=1; i<=n; ++i) {
      sum += term
      term += 2
   }
   return sum
}

BEGIN {
   n = 10
   sum = 0
   for (i=1; i<=n; ++i) { sum += oddsum(i) }
   print sum
}
```

Output

```
oddsum(1) called
oddsum(2) called
oddsum(3) called
oddsum(4) called
oddsum(5) called
oddsum(6) called
oddsum(7) called
oddsum(8) called
oddsum(9) called
oddsum(10) called
385
```

Writing to files: Use redirection

avg.awk

```
#!/usr/bin/gawk -f

BEGIN { FS = ":" }

{
   if ($2 == "sauropod") {
      pos = index($6,"-")
      if (pos == 0) {
         ts = te = int($6)
      } else {
         ts = int(substr($6, 1, pos-1))
         te = int(substr($6, pos+1, length($6)-pos))
      }
      if ((ts <= 252) && (te >= 201)) { nt++; sumt += $3 }
      else if ((ts <= 201) && (te >= 145)) { nj++; sumj += $3 }
      else if ((ts <= 145) && (te >= 65)) { nc++; sumc += $3 }
      else printf("Period cannot be determined for %s (%d,%d)\n", $1, ts, te);
   }
}

END {
   printf("Average lengths of sauropod dinosaurs\n") > "avg.txt"
   printf("   Triassic period (252-201 Ma)   : %6.2f meters\n", sumt / nt) >> "avg.txt"
   printf("   Jurassic period (201-145 Ma)   : %6.2f meters\n", sumj / nj) >> "avg.txt"
   printf("   Cretaceous period (145-65 Ma) : %6.2f meters\n", sumc / nc) >> "avg.txt"
}
```

The output

```
$ ./avg.awk dinosaurs.txt
Period cannot be determined for Brachiosaurus (155,140)
Period cannot be determined for Kotasaurus (205,190)
$ cat avg.txt
Average lengths of sauropod dinosaurs
   Triassic period (252-201 Ma)  :   4.88 meters
   Jurassic period (201-145 Ma)  :  20.60 meters
   Cretaceous period (145-65 Ma) :  18.40 meters
$
```

Notes

- > means overwrite.
- >> means append.
- The mode is determined by the first print statement.
- After that, there is no distinction between > and >>.
- The output filename is to be quoted, otherwise this would be treated as a variable.

Practice exercises

The exercises on this page use the dinosaur database given in the slides.

1. The program `average.awk` discards the records of sauropod dinosaurs living across multiple periods. Modify the program so that these records too are processed. For example, Kotasaurus lived in both the Triassic and the Jurassic periods, so use Kotasaurus in the statistics of both these periods.
2. Modify the program `theropod.awk` so that the printing is sorted with respect to the country names.
3. Write a gawk program `dinotypes.awk` that categorizes the dinosaurs in respect of their types. It should print the different types, and below each type, it should print the dinosaur names belonging to that type, one name in one line beginning with a tab. After the printing for each type, the program should print the count of dinosaurs of that type. For example, you should have a part of the output as follows.

```
ceratopsian:
        Albertaceratops
        Avaceratops
        ...
        Triceratops
Total count = 7
```

4. Write a program `sortsauropods.awk` to print the sauropod dinosaurs along with their living times and lengths, sorted with respect to the living times (earliest first). If there is a period (like 70–65 for Alamosaurus), use the average of the two endpoints as the living date of that dinosaur (67.5 for Alamosaurus).

Practice exercises

5. Study the format of the file `/etc/passwd`. Write a gawk script to find all the non-system users. For Ubuntu, these users have IDs $\geqslant$ 1000 (but not "too large").
6. Use the file `/etc/passwd` to let gawk categorize users with respect to their login shells.
7. A file `foomarks.txt` has the following format. The first line contains a single number (a positive integer) n standing for the number of tests conducted in the course FOO. The second line contains the list of maximum marks for the n tests. The third line contains the list of percentage contribution of the tests (it is the teacher's duty to ensure that the percentages add to 100). From the fourth line to the end, the file `foomarks.txt` stores the performance records of students. Each line consists of the following fields: student name, student roll number, mark in test_1, mark in test_2, $\ldots$, mark in test_n. Assume that all the students appeared for all the tests, so that all mark entries store numbers. Assume that each mark is a non-negative integer. Write a gawk script `computetotal.awk` that computes the overall performance of each student (in 100). Write the output to a file `foototal.txt` storing only student names, roll numbers, and total marks. Use tab as the field separator in the both the input and the output files.
8. Repeat the last exercise with the following exception. A mark entry may be `ABSENT`, indicating that the student did not appear for that test. Replace that string by 0 for calculating the total.
9. Repeat the last exercise with an additional exception. A mark entry may be `GRANT` indicating that the student was absent in that test, but the absence is granted for some genuine reason. Use the marks of other tests and scale the sum to 100 for calculating the totals in the presence of `GRANT` (and `ABSENT`).

Practice exercises

10. Write a gawk script `foogrades.awk` that reads the file `foototal.txt` produced by the last three exercises, and prints the students in the decreasing order of their total marks. In case of a match in the total, sort in the ascending order of roll numbers (unique for each student). The gawk script should also calculate the grade for each student under the absolute grading scheme (EX for [90, 100], A for [80, 90), and so on). **Rewrite** the input file `foototal.txt` by appending the grade of the student in each line.
11. Explain how gawk can process a `csv` file using comma as the separator and with strings double-quoted.
12. Write a gawk script `listall.awk` that reads a set of positive integers from the user, and prints the integers in the sorted order. The reading loop stops when the user enters 0 or a negative integer. Implement the list using (a) an array, and (b) an associative array. Duplicate entries by the user should be ignored.
13. Devise a scheme to store a binary min-heap (of positive integers) using the gawk programming language. Write the functions `init`, `heapify`, `insert`, `findmin`, and `deletemin` for min-heaps.
14. Devise a scheme to store a binary search tree (of positive integers) using the gawk programming language. Write the functions `init`, `insert`, `search`, and `inorderprint` for binary search trees.
15. A directed graph G is stored in a text file in the following format. The first line stores the number n of vertices, and the second line the number m of edges. This is followed by m lines each storing u, v for an edge (u, v) of G (assume that there are no duplicate entries for any edge). Write a gawk script that reads the graph file, reads two vertices u, v from the user, and determines (and prints) whether there is a u-to-v path in the graph. Number the vertices as $1, 2, 3, \ldots, n$.