$ cp student.lst rstudent.lst

Add following column at the end of rstudent.lst

35	20
36	21
37	22
38	23
38	24
40	25
1	5
2	6
3	7
4	8
5	9
6	10

$ cp student.lst wstudent.lst

1111	Amit B. Shah	01/01/1980	Surat	bca	9429123423
1112	Ramesh S. Chaudhari	20/02/1982	Baroda	bba	9926412126
1113	Sunil C. Makwana	11/03/1981	Vapi	bca	9925421213
1114	Samir P. Chowdhari	12/02/1982	Surat	bcom	8461626364
1115	Vimal K. Patel	18/03/1984			
2111	Bimal R. Surati	10/08/1990	Navsari	ba	9223423456
2112	Anil A. Joshi	15/02/1985	Valsad	bba	9712312342
2223	Ronak S. Surti	14/08/1970	Surat	bba	9595223231
2234	Jinal B. Chaudhri	17/07/1977	Bharuch	bca	9427194271
3122	Nilesh K. Makvana	25/03/1976	Bharuch	bca	9712323456
3123	Samira S. Chodhari	17/07/1977	Bharuch	bcom	9712323456
3124	Rita K. Makhwana	19/09/1976			

Create shortstudent.lst without spaces

$ tr -s ‘ ‘ <student.lst > shortstudent.lst

awk – An Advanced Filter

awk operates at the field level and can easily access, transform and format individual fields in a line. It also accepts extended regular expressions (ERE).

Syntax

awk options 'selection_criteria {action}' file(s)

The selection criteria is searching pattern, it filters data and select lines from file(s). The action criteria is action to perform on this line. The selection is done on one or more files.

Example

To display bca records from student.lst

```
$ awk '/bca/ {print}' student.lst
```

1111	Amit B. Shah	01/01/1980	Surat	bca	9429123423
1113	Sunil C. Makwana	11/03/1981	Vapi	bca	9925421213
1115	Vimal K. Patel	18/03/1984	Ahmedabad	bca	9526126126
2234	Jinal B. Chaudhri	17/07/1977	Bharuch	bca	9427194271
3122	Nilesh K. Makvana	25/03/1976	Bharuch	bca	9712323456

The above print bca records.

If selection_criteria is missing, the action applies to all lines. If action is missing, the entire line is printed. Either of the two (but not both) is optional, but they must enclosed within a pair of single (not double) quotes.

```
$ awk '/bca/' student.lst                print is default action
$ awk '/bca/ { print }' student.lst      whitespace permitted
$ awk '/bca/ {print $0}' student.lst     $0 is the complete line
```

awk also use regular expression BRE and ERE for pattern matching.

```
$ awk  -f  "|"  '/Sur[a]*ti/' student.lst
```

2111	**Bimal R. Surati**	**10/08/1990**	**Navsari**	**ba**	**9223423456**
2223	**Ronak S. Surti**	**14/08/1970**	**Surat**	**bba**	**9595223231**

Splitting a line into fields

awk uses the special parameter $0, to indicate the entire line. It identifies fields by $1, $2,$3……

By default, awk uses a contiguous sequence of spaces and tabs as a single delimeter. To specify other separator use the –F option.

Example

Display Name, City, Faculty and Mobile No. from Student.lst

```
$ awk  -F "|"  '/Surat/  {  print  $2,$3,$4,$6 }' student.lst
```

Amit B. Shah	**Surat**	**bca**	**9429123423**
Samir P. Chowdhari	**Surat**	**bcom**	**8461626364**
Ronak S. Surti	**Surat**	**bba**	**9595223231**

awk use NR built in variable to specify the line numbers.

Example

Display line 3 to 6 from student.lst

```
$ awk  -F "|"  'NR == 3,  NR == 6  { print  NR,  $2,$4}' student.lst
```

Sunil C. Makwana	**Vapi**
Samir P. Chowdhari	**Surat**
Vimal K. Patel	**Ahmedabad**

Formatted output (printf)

awk uses printf to format the data. awk accepts most of the formats used by printf function of C.

```
$ awk   -F "|"  '/Surati/ {

>  printf "%2d %4d %-20s \n", NR, $1,$2 }' student.lst
```

Here line number id 3 character wide right justified, Roll no is printed 4 character wide right justified and name is printed 20 character wide left justified.

Redirecting Standard Output

Every print and printf statement can be redirected with the > and | symbol. The filename or command in which the output is redirected must enclosed within double quotes.

The following command sorts the output.

```
printf "%s %s\n", $2, $3 | "sort"
```

The following output will be redirected to file resident_list.

```
printf "%s %s\n", $2, $3 | resident_list
```

Variables and Expressions

awk also use an expression which comprise string, number and variables. awk does not have char, int, lon, double etc. primitive data type. Every expression can be interpreted either as string or number.

awk also allows the user defined variables but without declaring them. Variables are case sensitive. X is different then x. awk variable do not use $ either in assignment or evaluation.

X="5"

print x

A user defined variable needs no initialization. It is implicitly initialized to zero or a null string. awk has a mechanisam of identifying the type and initial value of a variable from its context.

Strings in awk are always double quoted and can contain any character. It can contain escape sequences, octal value and hex value. Octal value are preceded by \ and hex value preceded by \x.

x="\t\tBELL\7"

print x

it prints two tabes, the string BELL display and sounds a beep.

Concatenation

awk provides no operator for concatenating strings. Strings are concatenated by simply placing them side-by-side.

x=”sun” ; y=”com”

print x y

suncom

print x.y

sun.com

The following example demonstrate how awk makes automatic conversions when concatenating and adding variables.

x=”5” ; y=6 ; z=”A”

print x y	y converted to string, so it print 56
56	
print x + y	x converted to number, so it print 11
print y + z	z converted to numeric 0 ; print 6

Expression also have true and false values associated with them. Any nonempty string is true. Any positive number is also true.

If (x)

Is true, if x is a non null string or positive number.

The comparison operators

We can compare whether city should surat or baroda using $4, because it is fourth field in student.lst.

Display students of Surat and Baroda city.

```
$ awk -F "|" '$4 == "Surat" || $4 == "Baroda" {
> printf "%-20s %-15s\n", $2, $3 }' student.lst
```

Display students who are not of surat and of bca.

```
$ awk -F "|" '$4 != "Surat" && $5 != "bca" {
> printf "%-20s %-15s\n", $2, $3 }' student.lst
```

The Regular Expression Operators : (~ and !~)

awk is also used to match the regular expression pattern in a field. awk offers ~ operator to match and != operator to negate match. To match a string embedded in a field, you must use ~ instead of ==. To negate match use !~ instead of !=.

Example

Display Records of Makwana, Makvana, Makhwana and Surati,Surti.

```
awk -F "|" '$2 ~ /Makh*[vw]ana/ || $2 ~ /Sura*ti/' student.lst        Check only second field.
```

Example

Display students who is not of bba or bcom.

```
awk -F "|" '$5 !~ /bba|bcom/ ' student.lst
```

Example

The following command display record ba and bba faculty both instead of ba. Because ba is embedded in bba.

```
awk –F "|" '$5 ~ /ba/' student.lst
```

To match a string at the beginning of the field, precede the pattern by ^. Use $ for matching a pattern at the end of a field.

Example

Display only ba faculty records

```
awk –F "|" '$5 ~ /^ba/' student.lst
```

Number Comparison

awk can also handle numbers both integer and floating type and also make relational test on them.

```
awk –F "|" ' $1 > 2000 {

> printf "%3d %-20s\n", $1, $2 }' student.lst
```

You can also combine regular expression matching with numeric comparison.

Example

Display rollno is greater then 2000 and whose mobile no ends with 56.

```
awk -F "|" '$1>2000 && $6 ~ /56$/' student.lst
```

Table: operators

Operator	Significance
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to
>=	Greater than or equal to
>	Greater than
~	Matches a regual expression
!~	Does not match a regular expression

Number Processing

awk can perform computation on numbers using the arithmetic operators +, -,*,/ and % (modulas).

Example : Display Rollno, name, mark1, mark2 and total

```
$ awk –F "|" $5="bca" {

>  printf "$2d  %-20s  %2d  %2d %3d\n" , $1, $2, $7, $8, $7+$8 }' rstudent.lst
```

Variables

awk has a certain built in variable like NR and $0. awk also permits user defined variables of his/her choice. You can now print a serial number using user define variable cnt, and apply it to bca students whose roll number more than 2000.

```
$ awk  -F "|"  '$5=="bca"  && $1 > 2000 {

>  cnt = cnt  + 1

>printf "%3d %-20s $-10s", cnt, $1, $2 $5 }' newstudent.lst
```

The initial value of cnt was 0 (by default). So the first line assigned the number 1.

You can also write

```
++cnt

cnt++

printf  "%3d\n", ++cnt                                  Increment count before printing
```

Storing awk program in a file (-f option)

You can store the large awk program in separate files and provide them with the .awk extension for easier identification. Do not speicify the single quotes to enclose the awk program.

```
$ cat > bcaawk.awk

$5=="bca"  && $1 > 2000 {

>  cnt = cnt  + 1

>printf "%3d %-20s $-10s", cnt, $1, $2 $5 }

[ctrl –d ]
```

Now you can use –f filename option to obtain the same output.

```
awk –F "|" empawk.awk student.lst
```

The BEGIN and END Sections

If you want to print something before the first line, for example a heading then the BEGIN section can be used. The END section is useful in printing some totals etc..

Syntax

BEGIN { action }

END { action }

Example

Display student mark list of surat student rollno, name, mark1, mark2, total mark

File Name : empawk2.awk

```
BEGIN {
printf "\t\t Student Mark List of surat \n\n"
} $4 == "Surat" {
cnt++ ;
fintotal+=$7+$8
tot1+=$7
tot2+=$8
printf "%2d  %-20s %-10s %2d %2d  %3d", cnt, $1, $2, $4, $7, %8, $7+$8
}
END {
printf "\n\t The Total Mark is %4d  %4d  %5d \n" , tot1, tot2, fintotal
}
```

Use the awk file empawk2.awk on rstudent.lst file

```
$ awk   -F "|"  -f  empawk2.awk  rstudent.lst
```

Use to perform calculation

Like other filters awk reads standard input when the filename is omitted. The following will do the floating point arithmetic.

```
$ awk 'BEGIN  { printf "%f\n", 22/7 }'

3.142857

[ctrl –d ]
```

Built-in-variables

awk has several built in variables, they are also assigned automatically, it is also possible for user to reassign some of them.

FS

By default awk uses a contiguous space as default delimeter. FS is used to give different separator like "|" which is generally used in database. The FS option must occur in BEGIN section so that body of the progam know its value. This is the alternname of -F option.

```
BEGIN  {  FS="|"  }
```

OFS

When you use print statement with comma separated arguments. Then each of this argument separated by space in output. Space is awk's default output field separator. The different output field separator is given by OFS variable in BEGIN section.

```
BEGIN { OFS="~"
```

NF

It is useful for cleaning up the database of lines that does not contain right number of field. By using it on wstudent.lst, you can locate those line not having six fields.

```
$ awk ' BEGIN  { FS  =  "|"  OFS="~"}

>  NF  !=  6

> print "Record no", NR, "has", NF, "Fields" }' wstudent.lst
```

FILENAME

FILENAME stores the name of the current file being processed. By default awk does not print the file name, but you can print it using the variable FILENAME.

```
awk  'city == "Surat"  {  print  FILENAME,  $0 }' student.lst
```

Table : Built-in Variable

Variable	Function
NR	Display line numbers of Records
FS	Input field separator
OFS	Output field separator
NF	Number of fields in current line
FILENAME	Current input file
ARGC	Number of argument in command line
ARGV	List of argments

Arrays

An array is a variable which can store a set of values.

An array in awk is different from other programming language in following respect:

- They are not formally defined. When it is used first time, it is considered as declared.
- Array elements are initialized to zero or an empty string unless initialized explicitely.
- Arrays expand automatically.
- The index can be virtually anything. It can even be a string.

Associative (Hash) Arrays

awk arrays are associative, where information is held as key-value pairs. The index is a key that saved internally as a string.

When we set day[1]=”Monday” it convert the index number 1 to string. There is no specific order in which the array elements are stored. The index 1 is different from 01.

```
$ awk ‘BEGIN {
> side[“N”] = “North” ; side[“S”] = “South” ;
> side[“E”] = “East” ; side[“W”] = “West” ;
> printf(“N is %s and E is %s\n”, side[“N”], side[“E”]);
>
> val[1] = 10 ; val[“1”] =30 ; val[“01”]=50 ;
> printf(“ val[1] is %s\n” val[1]);
> printf(“ val[01] is %s\n” val[01]);
> printf(“ val[\”1\”] is %s\n” val[“1”]);
>printf(“ val[\”01\”] is %s\n” val[“01”]);
> }’
```

Output:

N is North and E is East

val[1] is 10

val[01] is 10

val[“1”] is 30

The Environment Array (ENVIRON[])

awk maintains the associative array, ENVIRON[], to store all environment variables, like username, home directory etc…

```
$ awk 'BEGIN  {
>  print   "HOME = "  ENVIRON["HOME"]
> print   "PATH = "    ENVIRON["PATH"]
>}'
```

Output :

```
HOME = /home/students
PATH= /user/bin::/usr/local/bin:/usr/ccs/bin
```

Functions

awk has several built in functions that perform both arithmetic and string operations.

length

length determines the length of its argument, and if no argument is present, the entire line is assumed to be the argument.

You can use line without argument to locate lines whose length is exceed 40 characters.

```
awk  –F  "|"  'length  >  40' wstudent.lst
```

You can also use length with field.

Display the student who has short names.

```
$ awk  -F "|"  'length($2)  < 13' shortstudent.lst
```

index

Index (s1,s2) determines the position of a string s2 within a larger string s1.

```
$ awk  'BEGIN { printf " position of b in abcde is %2d", x = index("abcde","b") }'
2
[ctrl-d]
```

substr

substr(stg,m,n) function extracts a substring from a string stg.

m represent the starting point of extraction

n represent the number of characters to be extracted.

Display student records who born beween 1975 and 1985.

```
$ awk –F "|" 'substr($3,7,4) >= 1975 && substr($3,7,4) <= 1985' student.lst
```

split

split(stg,arr,ch) breaks up a string stg on the delimeter ch and stores the fields in an array arr[].

Display the date in student.lst in the format YYYY:MM:DD

```
$ awk -F \| '{split($3,ar,"/") ; print "ar[3]:ar[2]:ar[1]}' student.lst
```

system

system function is used to print system date, clear the screen…

```
$ awk –F "|" BEGIN {
> system("tput clear")
> system("date") }
> $4 == "Surat" { print $0 }
> END {
> printf "Records are finished"
> }' student.lst
```

Control flow Statement (if)

The if statement is a control statement. It execute set of statement when the condition of **if** is true.

Syntax

If (condition is true) {

Statements

} else {

Statements

}

If there is only one statement to execute, there is no need of curly braces in control flow constructs. If there are multiple statement to execute, the statements must be enclose in curly braces.

Display student records whose Rollno greater then 2000.

With following the condition return in selection criteria.

```
$ awk –F "|" ' $1 > 2000 { printf ($0) }' student.lst
```

OR

An alternative form is using **if**, but it require to place the if condition in action component.

```
$ awk –F "|" '{ if ( $1 > 2000 ) printf ($0) }' student.lst
```

Example

if (NR >= 3 && NR <=6)

if ($4 == "Surat" || $4 == "Baroda")

if ($5 ~ /^ba/)

if ($2 !~ /Sura*ti/)

Looping (for)

awk support both for and while loop. They both execute as long as control variable/command return a true value.

Example

Simple for program that display 1 to 5

```
awk 'BEGIN{

i=1

while(i<=5)

{print i

i++

}}'
```

Using for with an associative array

The for statement also use with an associative array. The loop select each index of an array.

syntax

for (k in array)

commands

Here k is the subscript of the array arr. Because k can also be a string.

Example 1:

Display all environment variables using for loop.

```
$ awk 'BEGIN {
> for ( key in ENVIRON )
> print key "=" ENVIRON[key]
>'
```

Example 2

Count student faculty wise from student.lst

```
$ awk -F "|" '{ cnt[$5]++ }
> END { for (faculty in cnt )
> print faculty, cnt[faculty] }' student.lst
```

```
bca     5
bba     3
bcom    2
ba      2
```

Looping with while

The while loop execute a set of statement as loon as control command of while is true.

Display 1 to 5

```
awk 'BEGIN{
i=1
while(i<=5)
{print i
i++
}}'
```