awk Programming

- **awk** script is a file with **comments** and **awk statements**, the comments are preceded by a **#** sign

```
# My awk assignment
/Jones/{print NR, $0}
/Tom/{print "Tom's birthday is " $4}
#End of Script
```

- to run **awk** with **awk** script

```
awk -f my_awk.txt employee.txt
cat employee.txt |awk -f my_awk.txt
```

  - Each line of the input file is read into awk's buffer **$0**
  - All the cmds in the script file are processed on this record
  - Then, the record is discarded, and the next line is read into **$0**.

AWK Variables

- Built-in Variables, more listed in Table 12.5 on p346

Name	Significance
NR	Number of Record of the being processed line(record)
NF	Number of Field of the being processed line (record)
$0	Entire record (the whole input line/record)
$1 - $n	Fields in the current record
FS	Record field separator, white space/tab by default
OFS	Output field separator
ORS	Output record separator (new line by default)
FILENAME	Name of the current input file

- User-defined variables
  - String, number or both
  - Declared on use, no need to declare it before using it
  - The type of the variable is determined by the value assigned
  - No prefix of "$" for variables except the pre-defined fields

```
# This is a comment line
# awk script file name is "script.awk"
BEGIN{
      print " This is the BEGIN block"
       count=0; FS=":"
}
/^north/{count++; print $1, $2, $3}
/^south/{count++; print "The", $1, " district."}
END{
      print "This is the END block"
      print "Number of records found:", count
}
```

Execution using the awk script

```
awk -f script.awk datafile
```

Programming with AWK

Conditional Statement (p350)

```
{  if (expression1){
     statement1; statement2;…
   }
   else if (expression2) {
      statement1
      statement2
    }
   else{
      statement
   }
}
```

- `if/else if/else`

awk '{if($6>50) print $1 " too high"; else \
print "Range is OK"}' file.txt

- Control statements
  - `exit`
    - Terminate the `awk` program
  - `next`
    - Read and process the next line

Program Control Statements

```
{if ( NR == 7) { exit(0)}}
{
    if ( NR == 5) { next}
    else {print $0}
}
```

Loop (351)

```
{   for (x=1; x<=NF; x++)
    {
        if ($x <0)
          print "Bottomed out!"; break
    }
    for (x=1; x<=NF; x++)
    {
        if ($x==0)
          print "Get next item"; continue
    }
}
```

```
{i = 1; while(i<=NF) { sum +=$i; print NF,$i; i++ }}
```

```
awk '{i=1; while (i<=NF) {print NF, $i; i++}}' file
awk '{for (i=1; i<= NF; i++) {print NF, $i}}' file
```

awk Built-in Functions (p348)

- **sub** and **gsub**

```
{sub(/PC/, "Personal Computer"); print}
```

  - sub: Substitute "PC" with "Personal Computer" on the first occurrence of PC on that line
  - If gsub is used, then global substitution, replacing every occurrence of the pattern

- **substr: substr(string, start_pos, length)**

```
echo today is |awk '{print substr($0, 2,2)}' ➔ od
```

- **split: split(string, array, field separator)**
  - The array subscript starts at 1

```
awk '{split($4,date,"/");if(date[3]>=60) print $0}' employee.txt
```

Tom Jones 4421 5/12/66 543354
Mary Adams 5346 11/4/63 28765
Sally Chang 1654 7/22/54 65000
Billy Black 1683 9/23/44 336500

Other approach?

More awk built-in Functions

- **`index`** function
  - Returns the first position where a substring is found in a string
  - Offset starts at position 1

```
{print index("hollow", "low")} → 4
awk '{print index($2,"Jones")}' employee.txt
```

- **`length`** function
  - Returns the number of characters in a string

```
length("hello") → return 5
{if(length($1) == 3) print $0}
```

- awk Built-In Arithmetic Functions
  - `int(x), log(x), exp(x), rand(), sqrt(x), srand(x)`
  - `atan2(x,y), cos(x), sin(x);`

Associative Array

- Regular array (numerical indexed array)
  - Indexed by number (integer)
  - `name[1], name[2]`
- Associate array (hash array, like mapping in c++)
  - Arbitrary index, called *key*

    `usage["Smith"], usage["Jones"],usage["Lin"].`
  - You need to use **double quotes** for the "key", otherwise, awk will treat the key as variable which may or may not have a value
- No need to declare arrays or index types
- A special **for loop** usage for associate array

```
for (key in my_array)
    print key, my_array[key];
```

Q:

How to get the totals of each person from the data set?

```
Susan 400
John 100
Mary 200
John 100
Mary 300
Susan 100
```

Approach

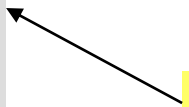
```
Susan 400
John 100
Mary 200
John 100
Mary 300
Susan 100
```

```
# file "ex.awk"
BEGIN{}

/^[A-Z]/{salary[$1] += $2} # avoid
empty/blank line

END{
      for (name in salary)
        print name, salary[name]
}
```

```
awk –f ex.awk data.txt
```

```
Output:
Mary 500
John 200
Susan 500
```

```
Susan:400
John:100
Mary:200
John:100
Mary:300
Susan:100
```



```
awk –F: –f ex.awk data.txt
```

```
# awk script file "employee.awk"
BEGIN{ print "Compute Average Salary"
       sum = 0
}
{salary[$1] = $5; sum += $5}
END {
       print "Average salary is:", sum/NR
       for (name in salary)
       {
              print name, salary[name]
              sum2 += salary[name]
       }
       print "Average salary is:", sum2/NR
}
```

```
Average salary is: 45437.5
Tom 54335
Sally 65000
Mary 28765
Billy 33650
Average salary is: 45437.5
```