AWK (I)

- AWK is a *UNIX programming language* used for manipulating data and generating reports
- AWK stands for the initials of the three authors
  - Alfred Aho, Peter Weinberger, and Brian Kernighan
- AWK scans input (file/stdin) line by line (as SED does)
  - searching for lines matching a specified pattern
  - performing specified actions by instructions enclosed by { … }
- Programming
  - Built-in functions, math functions, etc.
  - If statement
  - For/while, etc.

awk Basic Usage

- AWK command format (use single quotes, double quotes will be used inside the awk command)
  - `awk '/pattern/{action}' InputFile`
    - The action will be performed on lines which match the pattern
  - `awk '/pattern/' InputFile`
    - Print (default action) lines that matches the pattern
  - `awk '{action}' InputFile`
    - The action will be performed on **all lines**
- **awk** commands in script (file)
  - `awk –f script.awk InputFile`
- Input can be from file or STDIN, or pipe (output stream of previous command)
  - `who |awk '{print $1}'`
    - print out 1st field (user name) of the output from `who`

Pattern & Actions

- Pattern
  - Including regex, enclosed with forward slashes /…/

```
awk '/Tom/' employee.txt
awk '/Mary/{print $1, $3}' employee.txt
```

  - Be aware of the unsupported metacharacters, such as

```
awk '/[0-9]\{2\}/' file.txt (not work)
sed -n '/[0-9]\{2\}/p' file.txt (ok)
echo 123456 | awk '[1-9]{5}' (gawk, CS LAB)
```

- Actions
  - Action statements are enclosed with **curly brackets**
  - Can have multiple actions (statements) within the curly brackets
    - Actions are separated by semicolons on one line

```
'/Pattern/{ action statement one; action statement2}'
```

    - One action per line in script

```
/Pattern/{ 1st action statement
           2nd action statement
         }
```

      - No need to put semicolon at the end

How awk Works?

- Takes a line from input, assigned it to an internal variable $0
  - Breaks the line into fields/columns by white space or tab (default), and saves them in internal variables:$1, $2, up to the total number of fields $NF
  - Performs the actions on the line/fields if there are any. If no action is specified, default action is performed (line is printed to the screen)
- Takes next line from input file and puts to $0 and performs the actions on it, …until all the lines have been processed

Record & Fields

- Record: NR
  - Each line terminated with a newline is a record (be aware the difference of newline character on different platforms)
  - `$0:` is an internal variable, hold the entire record (whole line)
  - **`NR`**: number of records (number of lines) up-to-now
    - After a record (line) is processed, NR is incremented by one
- Fields: NF
  - Each record consists of fields separated by **field separator**, by default, it is either a whitespace or tab
  - NF: number of fields of each record, it can vary from line to line

```
awk '{print NR, NF, $0}' employee.txt
```

Input Field Separators

- For line in data.txt:

  Amy Lin|824-5164|UAH ITSC|AL 35899

  awk '{print $1, $2, NF}' info.txt → ???

  - Whitespace is the default separator
  - tab is treated as whitespace

  awk –F "|" '{print NF,":",$1, $2}' data.txt

  → 4:Amy Lin 824-5164

  - Now "|" is the input field separator

- Multiple field separators

  awk –F "[|]" '{print NF":"$1, $2, NF}' data.txt

  → 7:Amy Lin

  - Both "|" and " " are the input field separators

AWK Built-in Variables

- **FS**: Field Separator
  - White space: by default, can define your own with -F option
- **NF**: number of fields (each line) separated by **FS**
- **$n**: the nth field/column of a record (also called Positional Parameters)
- **OFS**: Output Field Separator
  - When print, {print $1, $2}, fields are separated by ",",which matches to a white space by default,

    OFS ⇔ a white space
  - You must use double quotes for "," if you want to print ","
- **NR**: internal variable: Number of Records

```
awk '{print NR, $1, $2 }' employee.txt
```

- More on P346 Table 12.5

Formatting Output

```
awk '/sally/{print NR, "\t\tHave a nice day," $1,\
  $2 "!"}' input.txt
```

- Here "t" is for tab space, a special character quoted by backslash
- Strings must be enclosed in double quotes to preserve the space in the print statement
- Commas (must) be used to separate the arguments
  - Comma is converted to whitespace (the default OFS)
  - If you want to print comma, it has to be enclosed in double quotes, or define OFS=","
- A new line **"\n"** is added by default for every print
  - If you don't want to start a new line, use **printf** instead of **print** for "fancy" output

Formatting Output

- The printf function
  - c-like function, formatting output

```
'{printf "The name is %-15s, ID is %8d\n", $1,$3}'
```

    - `%-15s` → left justified 15-space string
    - `%8d` → right justified 8-space integer

- See more on p334 of the textbook

awk Examples

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

- Retrieve specified record or fields of the record
  - **`awk '/Mary/' employee.txt`**
    - Prints lines containing "Mary"
  - **`awk '{print $1; print $4}' employee.txt`**
    - Prints the first and the fourth fields in two separated lines
  - **`awk '/^Sally/{print $1,$2}' employee.txt`**
    - Prints 1st and 2nd fields separated by a white space for lines starting with Sally
    - How to separate them ($1 & $2) with a tab?
    - How to accomplish this using other UNIX program?

Comparison & Logical Operators for line selection (p338, table 12.3)

- Number comparison: `<, <=, ==, !=, >=, >`
- String matching (regex): `~ , !~`

  "~" and !~ are used to match an expression within a field

```
awk '$2 ~ /Jones/' employee.txt
awk '$4 ~ /[6][0-9]$/{print $0}' employee.txt
awk '$4 ~ /\/[1-9][0-9]?\/[6][0-9]?/{print $2 ",", $1}'
  employee.txt
awk '$2 !~ /Jones/{print $0}'employee.txt
```

  - Spaces around `~` or `!~` are optional

- Examples

```
awk '$3 >= 124 {print NR, $0}' employee.txt
awk '$3 == 5346' employee.txt
awk '$1 !~ /Adam/{print NR, ":" $0}' employee.txt
awk '{max=($1 >$2)? $1: $2; print max}' input.txt
```

Mathematic operations in awk

- Arithmetic Operations (Table 12.2, p336)
  - +, -, *, /, %, ^

```
awk '$3*$4 > 500' input.txt
awk '{print NR, $3+10.99}' input.txt
awk '/southern/{print NR, $8/2}' input.txt
awk '/southern/{print NR, $8%2}' input.txt
```

  - awk does floating point operation
- Logical Operations (table 6.10, p191): &&, || , !

```
awk '$2 > 5 && $2< 15' inputFile
awk '$3 == 100 || $4 > 50' inputFile
awk '$3 == "Christ"{$3="Christian"; print}' datafile
awk '$3 ~ /Christ/{$3="Christian"; print}' datafile
(vs. sed -n 's/Christ/Christian/p' datafile)
```

More awk Examples

- `awk '/^[a-j]/ {print $1}' employee.txt`
- `awk '/[^A-Z]/ {print $0}' file.txt`
- `ls -l |awk '/hlin/ {print $4}'`
  - ⇔ `ls -l | awk '$2 ~ /hlin/ {print $4}'`
- `awk '/northeast/{print $3, $2}' datafile`
- `awk '/^[ns]/{print $1}' datafile`
- `awk '/^[A-Za-z]+/' datafile` ⇔
  `awk '/^[A-Za-z][A-Za-z]*/' datafile`
- `awk '$5 ~ /\.[7-9]+/' datafile`
- `awk '$2 !~ /E/{print $1, $2}' datafile`

BEGIN & END Blocks

- The **BEGIN** block
  - BEGIN Followed by an action block
  - The **BEGIN** block is executed before **awk** processes lines from the input file
  - **awk** does not start to read the input file until this action block has completed
  - Can be used to initialize variables, change the values of some default variables, such as FS, OFS, etc

```
awk 'BEGIN{FS=":"; OFS="\t"; ORS="\n\n"}{print $1 $2, $3}' file
```

- The **END** block
  - Followed by actions handled after all input lines have been processed
  - Can be used to do some statistics analysis on the input datafile:
    - sum, average, etc.

```
awk 'END {print "The total number of records is " NR}'
  input_file
awk '/Mary/{count++}END{print "Mary was found", count,
  \"times."}' input_file
```