# L'utility Unix `awk` [Aho-Weinberger-Kernighan]

L'utility **awk** serve per processare file di testo secondo un programma specificato dall'utente.

L'utility `awk` legge riga per riga i file ed esegue una o più **azioni** su tutte le linee che soddisfano certe **condizioni**. Azioni e condizioni sono descritte da un **programma**, la cui sintassi è simile al C.

Sinossi di `awk`:
`awk -Fc [-f filename] program` $\{$`variable=value`$\}^*$ $\{$`filename`$\}^*$

Ogni linea è vista come una sequenza di campi separati da `tab` e/o spazi. L'opzione `-F` serve per specificare un carattere separatore `c` sostitutivo.

`program` è un programma che specifica azioni e condizioni. Tale programma può comparire sulla linea di comando tra singoli apici oppure in un file (**awk script**) specificato con l'opzione `-f`.

Le variabili usate nel programma possono essere inizializzate sulla linea di comando.

Se sulla linea di comando non sono specificati file, allora `awk` legge l'input da std input.

# Programma/script awk (I)

Un programma awk consiste di una lista di uno o più comandi della forma:

`[ condition ] [ { action } ]`

dove

`condition` può essere:

- la condizione atomica `BEGIN`, verificata prima della lettura della prima linea in input;
- la condizione atomica `END`, verificata al termine della lettura dell'ultima linea in input;
- un'espressione contenente operatori logici e/o relazionali o una o più espressioni regolari.

Se la condizione non è specificata, allora l'azione viene eseguita su tutte le linee.

# Programma/script awk (II)

`action` è una lista di comandi tra i seguenti:

- `if` (cond) stat [ `else` stat ]
- `while` (cond) stat
- `do` stat `while` (cond)
- `for` (expr; cond; expr) stat
- `break`
- `continue`
- variable=expr
- `print` [ list of expr ] [ > expr ]
- `printf` format [, list of expr ] [ > expr ]
- `exit` *(salta alla riga di comando successiva)*

Se l'azione è omessa, viene eseguita l'azione di default, che è la stampa su standard output.

# Variabili

**Variabili predefinite**:

- \$1, \$2, ... contengono il $1^o, 2^o, \ldots$ campo della linea corrente;
- \$0 contiene l'intera linea corrente;
- `NF` contiene il numero dei campi della linea corrente;
- `NR` contiene il numero di linea corrente;
- `FILENAME` contiene il nome del file corrente;
- ...

**Variabili definite dall'utente**: non occorre dichiararle; sono automaticamente inizializzate a 0 oppure alla stringa vuota (a seconda dell'uso).

# Alcuni operatori e funzioni "built-in" di awk

- Operatori aritmetici:
  `+`, `-`, `*`, `/`, `%`, `^`, `++`, `--`
- Operatori logici:
  `!`, `&&`, `||`
- Operatori di confronto:
  `<`, `>`, `<=`, `>=`, `==`, `!=`
- Funzioni matematiche:
  `exp`, `log`, `sqrt`, `sin`,...
- Funzioni che operano su stringhe:
  `length`, `substr`, `index`, `tolower`, `toupper`,...
- Funzioni di "bit manipulation":
  `and`, `or`, `xor`, `compl`,...
- Funzioni che operano su data/ora:
  `mktime`, `strftime`

# Esempi (I)

Script awk che stampa il $1^o$, $3^o$ e ultimo campo di ogni linea:

```
BEGIN { getline; print "Start of file:", FILENAME }
      { print $1, $3, $NF }
END   { print "End of file" }
```

Se lo script awk è contenuto nel file `prog` e `testo` è un file di testo, possiamo eseguire il seguente comando:

```
> awk -f prog testo
```

```
  Start of file: testo
  campo1_linea_1 campo3_linea_1 ultimo_campo_linea1
  campo1_linea_2 campo3_linea2 ultimo_campo_linea2
  campo1_linea_3 campo3_linea_3 ultimo_campo_linea3
  End of file
```

Per stampare i campi 1,3 e ultimo solo delle righe 2 e 3, preceduti dal numero di linea:

```
>awk  'NR>1 && NR<4  { print NR, $1, $3, $NF }' testo
```

# Esempi (II)

Script awk (`prog1`) che conta il numero di linee e parole di un file, stampando su std output ciascuna linea con relativo numero:

```
BEGIN   { print "Scanning file" }
        { printf "line %d: %s\n", NR, $0;
          lineCount++;
          wordCount+=NF;
        }
END     { printf "lines=%d, words=%d\n", lineCount, wordCount }
```

```
> awk -f prog1 testo

  Scanning file
  line 1: campo1_linea_1 campo2_linea_1 campo3_linea_1 ultimo_campo_linea1
  line 2: campo1_linea_2 campo2_linea_2 ultimo_campo_linea2
  line 3: campo1_linea_3 campo2_linea_3 campo3_linea_3 ultimo_campo_linea3
  lines=3, words=11
```

# Esempi (III)

Il seguente script awk stampa i campi di ciascuna linea in ordine inverso:

```
{ for (i=NF; i>=1; i--)
      printf "%s", $i;
  printf "\n";
}
```

Il seguente script stampa le righe che contengono una `t` seguita da almeno un carattere e poi da una `e`. La condizione è espressa mediante un'**espressione regolare** racchiusa tra due /:

```
/t.+e/ { print $0 }
```

Una condizione può essere espressa da 2 espressioni regolari separate da ",". awk esegue l'azione corrispondente su tutte le linee comprese tra la prima linea che soddisfa la prima espressione alla successiva che soddisfa la seconda espressione. Esempio: `/strong/, /clear/ { print $0 }`

stampa tutte le linee comprese tra la prima linea che contiene la stringa `strong` e la successiva che contiene la stringa `clear`.

# Esempi (IV)

Il seguente comando fornisce su std output le linee di lunghezza maggiore di 10 caratteri del file `prova`:

```
> awk '{ if (length >10) print $0 }' prova
```

dove `length` è una funzione predefinita.

**Esercizio:** Scrivere un comando `awk` per stampare il numero massimo di campi di una linea in un dato file.

# I Progetto: La shell Unix

Si progetti uno script per la shell `bash` che accetta come parametri una sequenza di nomi di file (ordinari o directory) e li rimuova spostandoli nella cartella `.cestino` della propria home (invece di cancellarli fisicamente). Nel caso in cui la directory `.cestino` non esista lo script dovrà crearla.
Nel caso in cui i file da rimuovere abbiano dei nomi che corrispondono ad altri file già presenti in `.cestino`, lo script dovrà chiedere all'utente se desidera sovrascrivere questi ultimi o se desidera rinominare i file da spostare.

Si implementino inoltre le seguenti opzioni:

- `-l` che permetta di visualizzare i contenuti della directory `.cestino`;
- `-p` che svuoti la directory `.cestino`;
- `-r <nome file> <percorso>` che ripristini il file `<nome file>` spostandolo dalla directory `.cestino` a quella specificata da `<percorso>`. Nel caso in cui nella directory di destinazione esista già un file con lo stesso nome, lo script dovrà chiedere all'utente se desidera sovrascrivere quest'ultimo o se desidera rinominare il file da ripristinare.

Si gestiscano opportunamente gli errori.
Si commenti opportunamente il codice.
Si provi lo script su alcuni esempi significativi.