Linguaggi di progr. e variabili

- Nei linguaggi evoluti ogni variabile può contenere esclusivamente dati di un certo tipo, l'associazione fra variabile e tipo è fatta prima dell'uso
- Es:

```
float  num;
num = 3.14;
```

- Non posso eseguire:

```
num = “pippo”
```

Variabili

- Le variabili in AWK non sono tipate e non vengono dichiarate
- Una variabile esiste dal momento in cui è utilizzata x la prima volta come lato sinistro di un assegnamento:

  Vrb = espressione

- Può contenere stringhe di caratteri oppure valori numerici

Variabili

- Una variabile non ha un tipo permanente, può contenere in momenti diversi numeri o stringhe
- Se una stringa viene usata per errore in un'espressione aritmetica assume valore 0 (non viene segnalato un errore)
- Attenzione!!!

var1 contiene la stringa pippo ma ...

```
var1 = “pippo”
var1 = var1 + 3
```

... a questo punto var1 conterrà il numero 3

Usi leciti di variabili

- In AWK si possono fare cose strane ...

```
Due = 2;
Tre = 3;
print (Due Tre) + 4;
```

- Il risultato sarà 27!
- Una variabile AWK è il segnaposto di un valore (macro)

Concatenazione di stringhe

- Esempio:

Prefisso = “011”
Tel = “12345”
print “(“ prefisso “)” Tel
print “(“ prefisso “)” , Tel

- Ottengo:

(011)12345
(011) 12345

Operatori

- Oltre a + - / * (somma, sottrazione, divisione e moltiplicazione):

```
x ^ y  (x elevato a y)
x ** y  (x elevato a y)
x++    (incrementa x di una unità)
x--    (decrementa x di una unità)
x % y  (modulo, resto della divisione intera)
```

- Le stringhe possono solo essere con-catenate

Istruzioni

- Assegnamento
- If-else: esegue azioni diverse a seconda del valore di verità di una condizione
- While: ripete un'azione fintantoché la condizione è vera
- Do-while: ripete un'azione fintantoché una condizione è vera
- For: ripete per un certo numero di passi

Esempi

```
if ((x % 2) == 0)
   print “pari”
else
   print “dispari”

if ((x % 2) == 0)  print “pari”; else
   print “dispari”
```

Esempi

```
i = 1                          inizializzazione
while (i < 4) {                condizione
    print $i;
    i++;                       azione di progresso
}
```

```
i = 1                          inizializzazione
do {                           condizione
    print $i;
    i++;                       azione di progresso
} while (i < 4)
```

Esempi

```
for (iniz ; condiz ; incremento) {
      Istruzioni
}

for (i=1; i<=3; i++) {
      If (($i %2) == 0) cont_pari++;
}
```

Equivalenza for/while

```
for (i=1; i<=3; i++) {
	If (($i %2) == 0) cont_pari++;
}
```

- Equivale al ciclo while:

```
i=1;
while (i<=3) {
	If (($i %2) == 0) cont_pari++;
	i++;
}
```

True e False

- Non esiste il tipo di dato boolean
- Il numero 0 e la stringa vuota “” corrispondono a falso
- Tutto il resto corrisponde a vero
- Esempio:

```
if (“sono una stringa”)  print “vero”;
if (x = 31) print “vero”;
if (“0”) print “vero”;
```

Operatori relazionali

- `<` `<=` `>` `>=`: minore, minore uguale, maggiore, maggiore uguale
- `==`: uguale
- `!=`: diverso
- `~`: matching parziale
- `!~`: no matching
- *elem* in *array*: l'elemento è contenuto nell'array

Esempi

- (cont >= 15): vera se il valore della variabile cont è maggiore o uguale a 15
- ($2 == “pippo”): vera se il secondo campo ha per valore la stringa “pippo”
- (cont != 0): vera se il valore della variabile cont è diverso da 0
- ($3 ~ /ab*/): vera se il campo $3 ha un match-ing parziale con l'espressione regolare ab*
- ($3 !~ /ab*/): vera se non c'è alcun matching

Espressioni booleane

- cond1 && cond2: and, la condizione composta è vera se cond1 e cond2 sono vere
- cond1 || cond2: or, la condizione composta è vera se almeno una delle due sottocondizioni è vera
- ! cond: negazione, la condizione composta è vera se cond è falsa

Esempi

- ((cont >= 0) && (cont < 10)) : vera se il valore della variabile cont è nell'intervallo [0, 10[
- (($2 == “pippo”) || ($2 == “pluto”)): vera se il secondo campo ha per valore la stringa “pippo” oppure la stringa “pluto”
- (!cont): vera se il valore della variabile cont è falso
- !(E1 && E2) equivale a !E1 || !E2
- !(E1 || E2) equivale a !E1 && !E2

Esempio riassuntivo

```
BEGIN { both = 0; flag1 = 0; flag2 = 0; }
$1 ~ /(ab)+/  {
        flag1 = 1;
        }
$2 ~ /(ab)+/ {
        flag2 = 1;
        }
flag1 && flag2 {
        both++;
        }
{ flag1=0; flag2=0;}
END { print both }
```

Stampa il numero di record in cui sia il campo 1 sia il campo 2 hanno un matching con (ab)+

Provare su:

```
abab  abab
abab 111
111 ab
ab abab
```

Programmi equivalenti

```
BEGIN { both = 0; }
$1 ~ /(ab)+/  &&  $2 ~ /(ab)+/  {
        both++;
        }
END { print both }

BEGIN { both = 0; }
{
   if (($1 ~ /(ab)+/) && ($2 ~ /(ab)+/)) both++;
}
END { print both }
```

Esempio ...

- Supponiamo che il numero di campi non sia costante e che vogliamo visualizzare quali campi hanno un matching con (ab)+
- Occorre un ciclo while

... Esempio

```
$0 ~ /(ab)+/  {
        i=1;
        while (i <= NF) {
                if ($i ~ /(ab)+/) print "campo", i, $i
                i++;
        }
}
```

Variabili di "sistema"

- NF: numero di campi del record corrente
- NR: numero del record corrente
- RS: separatore di record (default: newline)
- FS: separatore di campo (default: spazio)

Es. programma per visualizzare le vrb. di sistema record x record

```
{
 print "NR: ", NR
 print "NF: ", NF
 print "RS: ", RS
 print "FS: ", FS
}
```

Esempio

```
1) BEGIN {FS="aaa"}
2) {
3)   print "record", NR
4)   i=1;
5)   while (i<=NF) {
6)     print "    ", $i;
7)     i++;
8)   }
9) }
```

dati

```
1234aaa3456aaa2321
1212aaabmbmbmbmaaag
llkkjjaaappooiiaaapojniaaa898
```

Risultato

record 1
1234
3456
2321
record 2
1212
bmbmbmbm
g
record 3
llkkjj
ppooii
pojni
898

dati

1234aaa3456aaa2321
1212aaabmbmbmbmaaag
llkkjjaaappooiiaaapojniaaa898

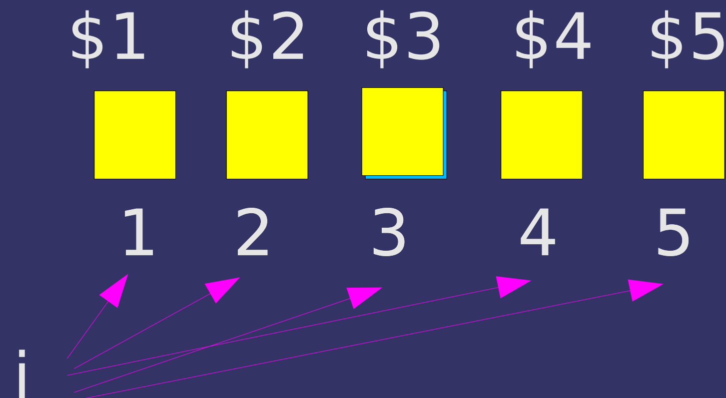
Analisi del codice

Regola eseguita 1 sola volta

```
1) BEGIN {FS="aaa"}
2) {
3)   print "record", NR
4)   i=1;
5)   while (i<=NF) {
6)     print "    ", $i;
7)     i++;
8)   }
9) }
```

Regola eseguita x ogni record: contiene 3 istruzioni, la III è un ciclo

Es. NF uguale a 5:



Esempio

1) BEGIN {FS="aaa"}
2) $2 ~ /b/ { print "record", NR, "campo", $2 }

dati

```
1234aaa3456aaa2321
1212aaabmbmbmbmaaag
llkkjjaaappooiiaaapojniaaa898
```

Risultato:

record 2 campo bmbmbmbm

Next

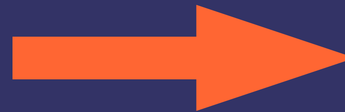
- L'istruzione speciale next interrompe l'elaborazione di un record e costringe awk a saltare direttamente a quello successivo
- Esempio:

```
$1 == "START" {interested = !interested; next}
$1 == "END"    {interested = !interested; next}
interested      { print $1 }
```

Cosa fa il programma?

11212121311
START
A b c
B c d
D e f
END
1213131111
1213131313
START
A b c
END

Dati di partenza



Risultato

A
B
D
A

Esempio, spiegazione

```
1) $1 == "START" {interested = !interested; next}
2) $1 == "END"    {interested = !interested; next}
3) interested     { print $0 }
```

- Supponiamo che nel file di dati venga prima un record che contiene START e dopo un po' un record che contiene END
- interested inizialmente vale 0, ovvero falso; alla prima occorrenza di un record che ha nel primo campo il valore START la regola 1 fa diventare interested vera
- Anche la terza regola è applicabile però l'istruzione next forza l'interruzione dell'analisi del record corrente