

# Espressioni condizionali e cicli

# L'*exit status*

- Ogni comando/programma restituisce un intero detto *exit status* al chiamante
- in C, è l'intero restituito dalla funzione main
- Di norma,
  - 0 = terminazione regolare
  - diverso da zero = terminazione irregolare
- La variabile di shell “?” contiene l'exit status dell'ultimo comando eseguito
  - Provare a eseguire un comando qualsiasi, e poi “echo \$?”

# Operatori su comandi

- `cmd1 ; cmd2`
  - esegue `cmd1` seguito da `cmd2`
- `cmd1 && cmd2`
  - esegue `cmd1`; poi, esegue `cmd2` se `cmd1` è terminato con successo (`exit(cmd1) == 0`)
- `cmd1 || cmd2`
  - esegue `cmd1`; poi, esegue `cmd2` se `cmd1` è terminato con errore (`exit(cmd1) != 0`)
- in tutti e tre i casi, l'exit status complessivo è quello dell'ultimo comando eseguito

# Comando *if*

```
if comando  
then  
    lista comandi  
[elif comando  
    lista comandi]  
[else  
    lista comandi]  
fi
```

- come test si usa l'exit status di un comando
- per mettere if e then sulla stessa linea, usare “:”

# Espressioni condizionali

- il comando “test exp” valuta exp come espressione condizionale
  - cioè, termina con exit status 0 se exp è vera
  - “test exp” si può abbreviare “[ exp ]” (spazi obbligatori)
- operatori ammessi:
  - su stringhe: ==, !=, -z
  - su interi: -lt, -le, -eq, -ne, -ge, -gt
  - operatori unari su nomi di file: -e, -f, -r, -w, -x
- per informazioni on-line: “man test”

# Espressioni condizionali

```
if [ -z "$1" ]  
then  
    echo "Questo script richiede un argomento."  
    exit 1  
fi
```

```
if [ $# -lt 4 ]  
then  
    echo "Questo script richiede 4 argomenti."  
    exit 1  
elif [ ! -e "$1" ]  
then  
    echo "Il file $1 non esiste."  
    exit 1  
fi
```

# Sostituzione aritmetica con `$(( ... ))`

- `$(( exp ))` valuta `exp` come espressione aritmetica
- `$(( exp ))` viene sostituito dalla shell con il valore di `exp`
- solo aritmetica su numeri interi
- Esempi: sia “a” una variabile con valore 7

espressione	sostituita con	note
<code>\$(( \$a+1 ))</code>	8	
<code>\$(( \$a++ ))</code>	8	“a” viene incrementata
<code>\$(( \$a*3 &gt; 8 ))</code>	1	1 equivale a “vero”

# Sostituzione aritmetica

- Operatori:
  - aritmetici: +, -, /, \*, %
  - elevamento a potenza: \*\*
  - bit-a-bit: <<, >>, &, |, ~
  - booleani: <, <=, ==, !=, >, >=, &&, ||, !
- Come si usa un'espressione aritmetica come espressione condizionale?

# Ciclo *while*

```
while comando  
do  
    blocco comandi  
done
```

**Esempio:**

```
i=0  
while [ $i -lt 10 ]  
do  
    i=$(( $i+1 ))  
done
```

ripete la lista di comandi fintantoché  
il comando viene eseguito con successo  
(come in C)

# Ciclo *while*

Esempio:

```
while true
do
    echo    Inserisci il nome di un file \
    da visualizzare (q per uscire):
    read nome_file
    if [ nome_file == q ]
    then
        break
    else
        cat $nome_file
    fi
done
```

# Ciclo *for*

```
for var in lista valori  
do  
    blocco comandi  
done
```

- *lista valori* è come una lista di argomenti passata a un comando
- Esempi:
  - for a in 1 2 3
  - for a in \$(ls)
  - for a in “uno” “due” “tre” (diverso da for a in “uno due tre”)
  - for a in “\$@” (diverso da for a in “\$”)
  - for a in \*.txt

# Esercizi

Il file elenco.txt contiene un elenco di studenti. Ciascuna riga e' nel formato: cognome,nome,matricola,indirizzo email.

Scrivere un comando per:

- 1) elencare tutti gli studenti che non appartengono naturalmente al gruppo I.
- 2) elencare tutti gli host che gli studenti usano per la posta, senza ripetizioni.
- 3) contare il numero di studenti che hanno un nome composto da almeno due parole (attenti agli spazi prima e dopo il nome).

# Esercizi

4) Script “*sw file1 file2*”: dati come argomenti due file esistenti, scambia i loro nomi.

5) Script “*bis comando*”: dato come argomento un comando, lo esegue due volte di seguito.

6) Script “*add\_comment*”: per ciascun file .c della directory corrente, aggiunge all'inizio del file una riga di commento che indica il proprietario del file e la data di ultima modifica