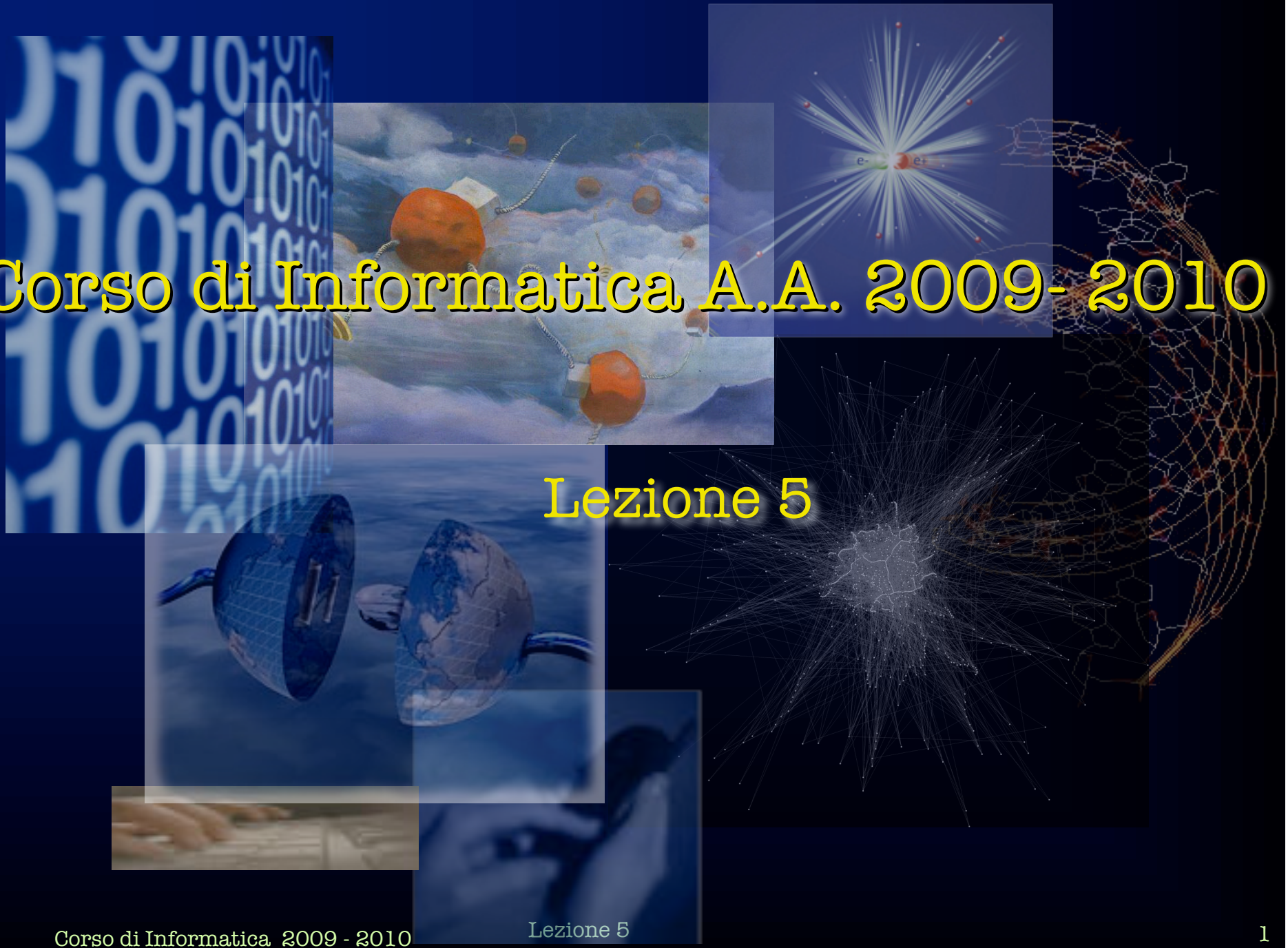


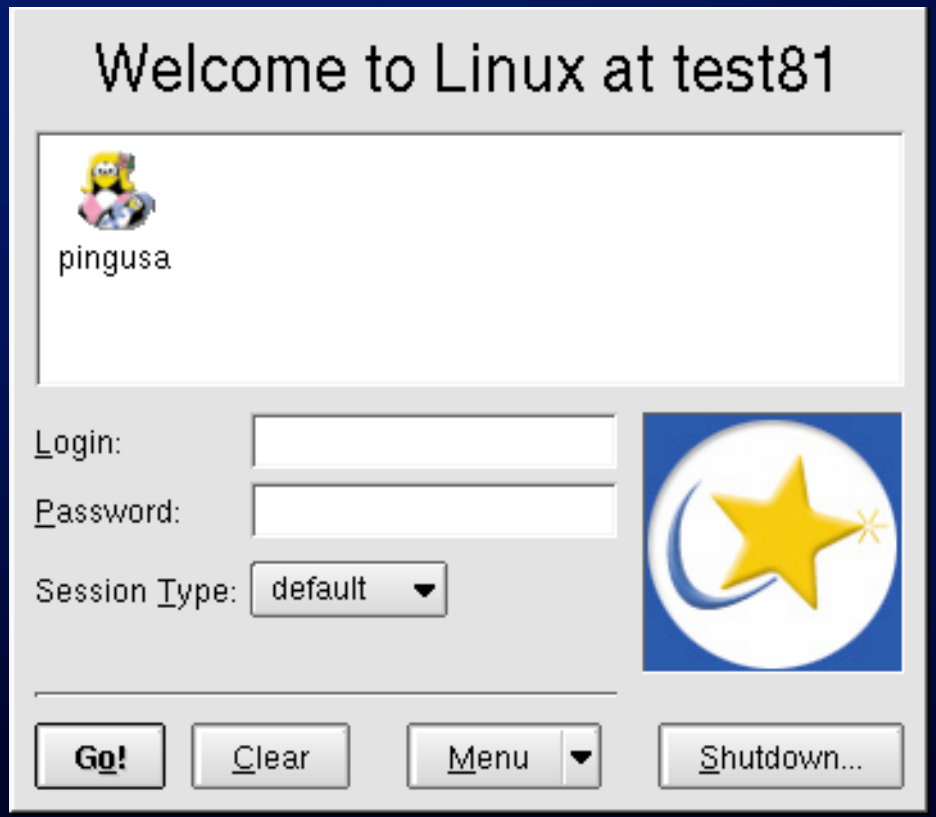
Corso di Informatica A.A. 2009-2010

Lezione 5

Linux per neofiti

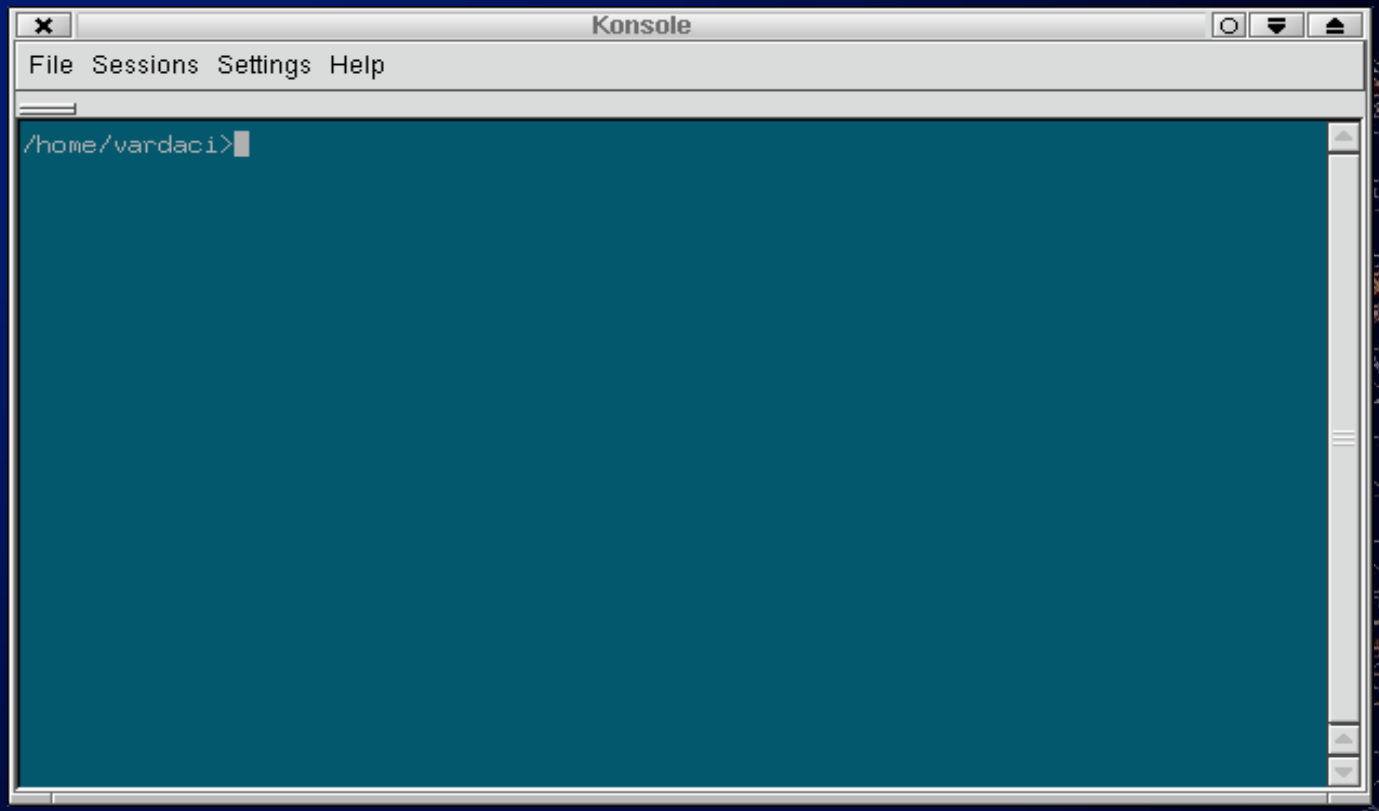
- Interazione shell-terminale
- Gestione della sicurezza
- Gestione della protezione
- Principali comandi Unix
- Redirezione di I/O
- Files di startup

….la prima volta… o quasi….



Login manager

Interazione shell-terminale (I)



Quando si apre una shell si entra in una directory di lavoro (default) che e' assegnata all'utente dal system administrator. In questo esempio e' /home/vardaci.

E' possibile navigare nel file system con il comando `cd`.

Gestione della Sicurezza

- ogni utente e' identificato da un ***account*** cui corrisponde un User ID (UID) e un Group id (GID)
- ogni utente si deve far riconoscere dal sistema per mezzo della combinazione ***username/password***

Gestione della Protezione (I)

- Nei sistemi monoutente la protezione e' limitata ai singoli files i quali possono essere protetti con una password;
- Nel sistema multiutente UNIX si definiscono tre classi di protezione user (u), group (g) e others (o) ed ad ognuna di esse e' associata una tipologia di modi accesso

Gestione della Protezione (II)

Le tipologie di modi di accesso sono tre:

r read
w write
x execute

Il proprietario del file (user) puo' impostare queste tre tipologie per il file stesso.

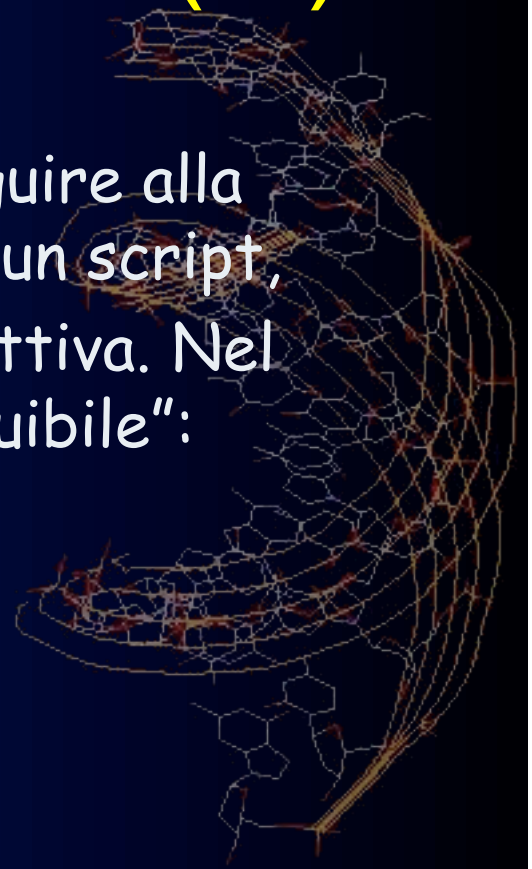
Ad es, il proprietario puo' negare la lettura del file ai membri di un gruppo e a chiunque altro, oppure fare in modo che altri utenti accedano ad un directory:

```
$ chmod go-r  expdata.dat
$ chmod o+x /home/sviluppo
```

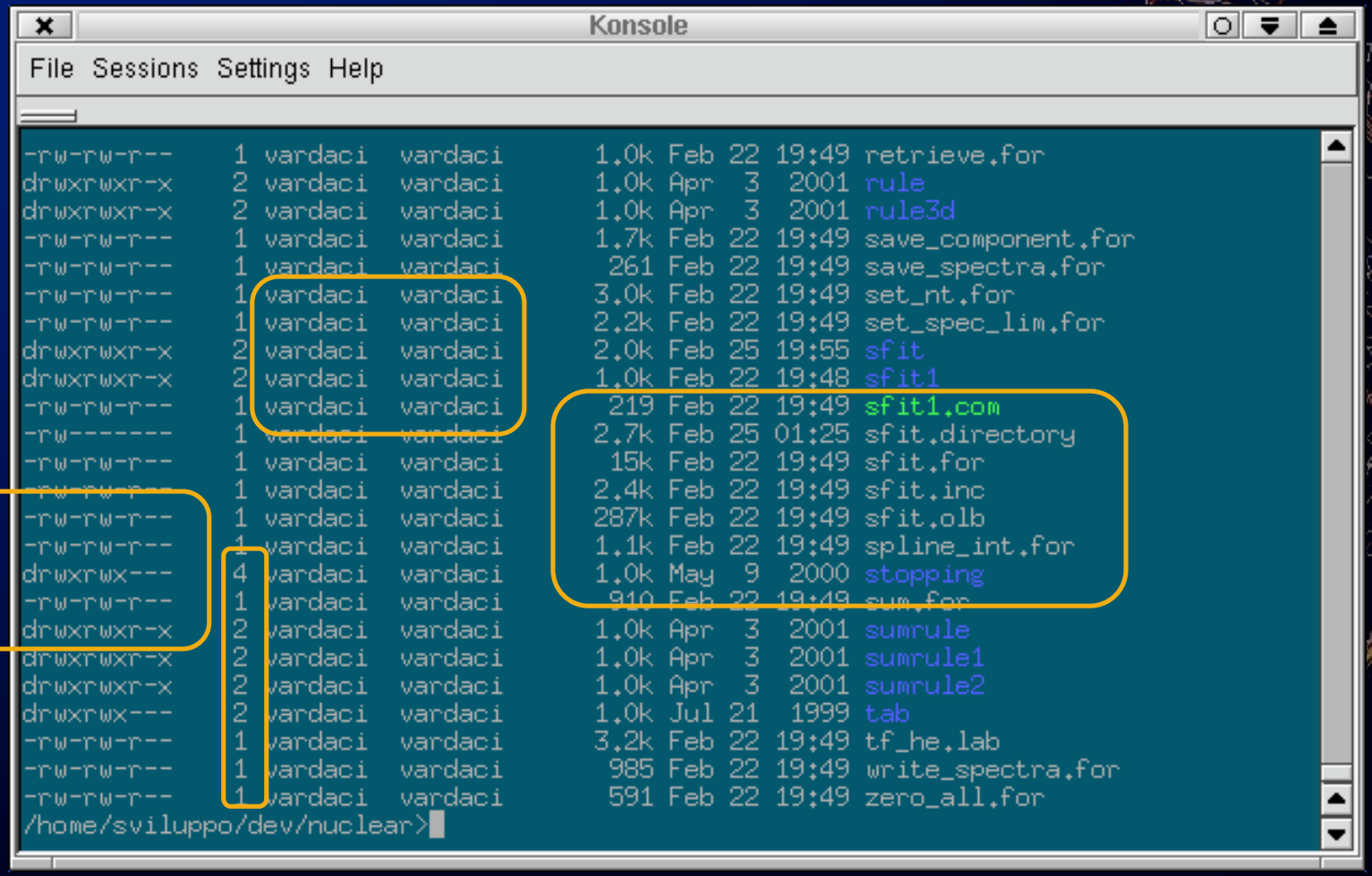
Gestione della Protezione (III)

Osserviamo che quando si voglia fare eseguire alla shell dei comandi scritti in un file, ovvero un script, la tipologia execute del file deve essere attiva. Nel gergo si dice che "il file deve essere eseguibile":

```
$ chmod ugo+x  commands
```



Gestione della Protezione (IV)



Gestione della Protezione (V)

le tre classi di protezione u,g ed o possono essere combinate con i tre tipi di accesso (read, write, execute) :

rwx	rwx	rwx
user	group	other

La presenza di un permesso e' indicata dalla lettera appropriata in una data locazione convenzionale.

Gestione della Protezione (VI)

Modo di Accesso	Files Ordinari	Directory Files
Read	Permette l'esame del contenuto	Permette di accedere all'elenco dei files nella directory
Write	Permette la modifica del file	Permette di creare o rimuovere files
Execute	Permette l'esecuzione degli scripts	Permette l'accesso alla directory

Navigazione nel file system

`cd nomedir`	Cambia la directory corrente
`pwd`	Mostra la directory corrente
`ls [-lh…] [nomedir]`	Elenca i files nella directory corrente o in nomedir
`df [-h][mountpoint]`	Disk free: stato di utilizzo di un disco
`du [-ak] [nomefile]`	Dimensioni di files e directory

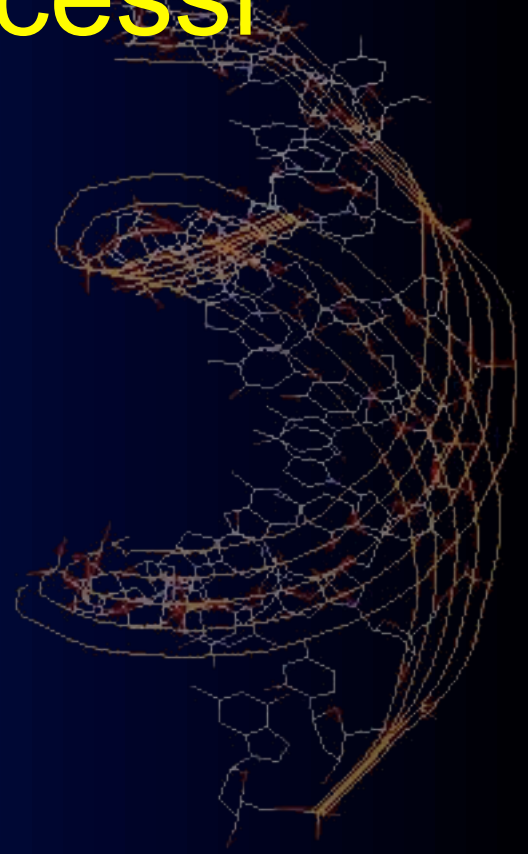
Manipolazione di files (e directories)

`cp`	`more`	`cat`
`rm`	`touch`	`diff`
`mv`	`head`	`sort`
`mkdir`	`tail`	`tar`
`rmdir`	`chmod`	`gzip`
`ln`	`chown`	`bzip2`
`file`	`chgrp`	`find`

Fate riferimento ai testi per i dettagli…

Manipolazione dei processi

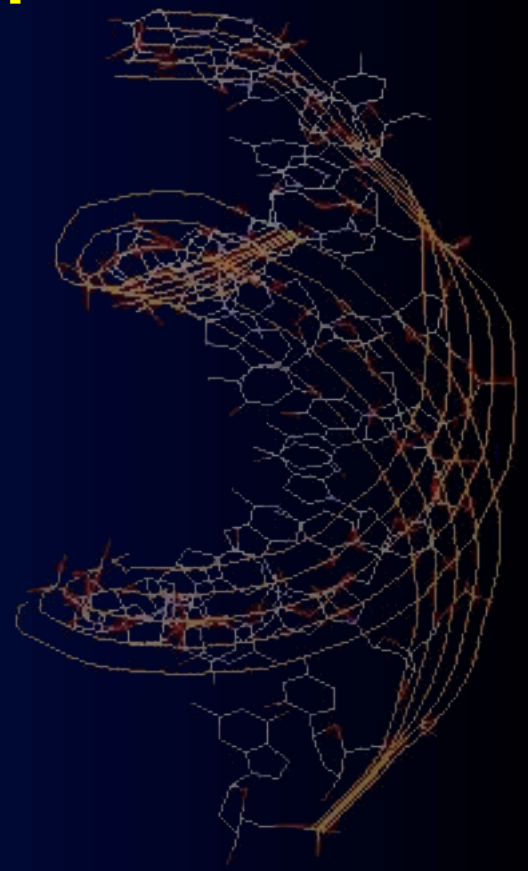
```
ps
bg   (oppure &)
fg
kill
top
nice
```



Comandi “generali”

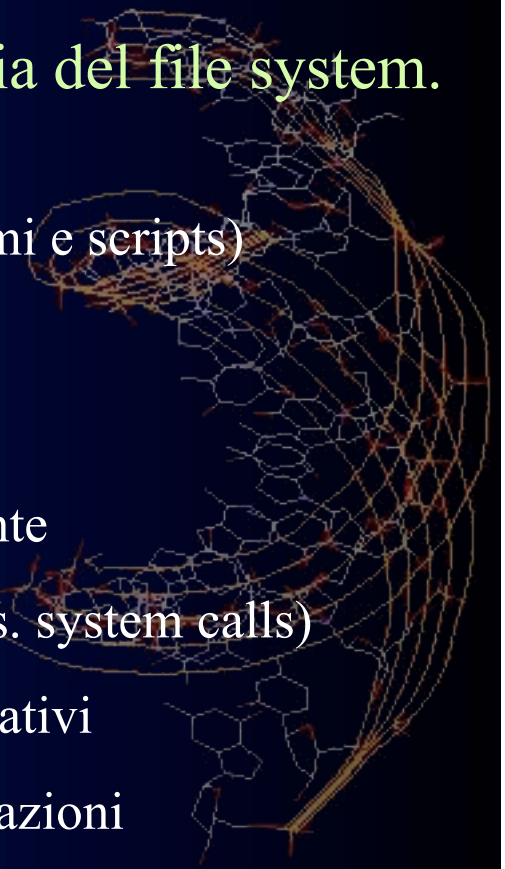
```
man
who
whoami
lpr
echo
alias
which
```

```
chsh
passwd
date
time
grep
uname
```



Principali directory e sottodirectory

La directory root (=radice) contiene l'intera gerarchia del file system.



/bin :	file eseguibili essenziali (bin = binary, programmi e scripts)
/dev :	file dei dispositivi di sistema (dev = device)
/etc :	file di configurazione caratteristici del sistema
/home :	directory relativa alle aree associate ad ogni utente
/lib :	librerie essenziali per la gestione del sistema (es. system calls)
/usr :	raccoglie la maggior parte dei programmi applicativi
/tmp :	contiene file temporanemente creati dalle applicazioni
/sbin :	file eseguibili necessari all'avvio del sistema
/mnt :	punti di mount per file systems temporanei (floppy,cdrom…)

Ridirezione dell' I/O (I)

Un qualunque processo apre automaticamente tre tipi di files: standard input, standard output e standard error.

Tipicamente questi file sono identificati con i numeri 0, 1 e 2, detti descrittori, e sono associati, rispettivamente, alla tastiera, al terminale video e ad una console di controllo.

Lo scopo della ridirezione e' quello di *ridirigere* questi tre canali di comunicazione verso altri sistemi di I/O, come ad es. un file di testo.

In UNIX, la ridirezione dell' I/O e' realizzata attraverso gli operatori < e >.

Ridirezione dell' I/O (II)

Es. Ridirigere la lista dei file di una directory dal terminale ad un file.

```
$ls /etc/init.d 1> /home/peppe/my_inetd
```

Lo *standard output* viene ridiretto (>) verso un file. Se il file non esiste viene creato al momento (on the fly); se esiste viene sovrascritto.

```
$ls /etc/init.d > /home/peppe/my_inetd
```

In questo caso la shell assume che vogliate usare lo *standard output*, ma fornisce un messaggio di errore se il file gia' esiste.

Ridirezione dell' I/O (III)

Es. Quante parole ci sono in un file?

```
$ wc -l 0< /home/peppe/my_inetd
$ wc -l < /home/peppe/my_inetd (default)
```

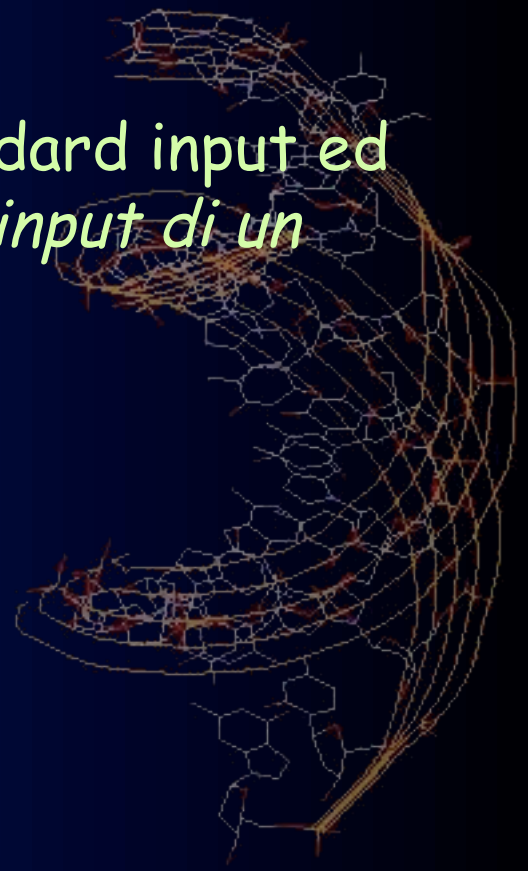
Spesso e' utile ridirigere lo *standard error* , ovvero il canale in cui I processi riversano i loro messaggi d'errore:

```
$ wc -l > /dev/null 2>     \
                /home/peppe/my_error
```

Pipe: comandi concatenati

La pipe permette di "concatenare" gli standard input ed output: *l'output di un programma diventa l'input di un altro programma con il simbolo* |.

```
$ cat my_file | lpr
```



Caratteri speciali: Metacaratteri

Un metacarattere serve a sostituire un insieme contiguo di uno o piu' caratteri.

Es. i caratteri * e ?

```
$ls *.txt
```

Questo comando permette di elencare solo i files I cui nomi che terminano con ".txt".

```
$ls g?.pdf
```

Questo comando permette di elencare solo i files i cui nomi sono di due lettere, iniziano con la lettera g e terminano con ".pdf". (ad es. g1.pdf ma non g12.pdf)

Il compito di individuare i files che corrispondono all'espressione *.txt non viene svolto dal comando `ls` ma dalla shell stessa.

Path

I comandi Unix, come detto sono dei particolari files eseguibili. Come fa l'interprete dei comandi a sapere dove cercare questi files ?

La risposta è nel valore di una particolare variabile chiamata path che contiene un elenco di directories in cui cercare un comando o un generico eseguibile (script, programma etc.). La path conterrà in genere un certo numero di directories "standard" come /bin, /usr/bin e ./

Altre directories possono essere aggiunte dall'utente per eseguire direttamente comandi particolari.

Quando si invoca un comando il sistema "sfoglia" le directories elencate nella path (nell'ordine) ed esegue il primo comando corrispondente.

Variabili di ambiente

La path è solo un esempio di una classe di variabili, dette variabili di ambiente, che vengono utilizzate dal sistema, o da particolari applicazioni per contenere informazioni specifiche della shell. Il comando echo, e il simbolo speciale $ permettono di visualizzare il contenuto di tali variabili. Ad esempio:

```
echo $path
```

Visualizza la path

```
echo $home
```

Visualizza la home directory

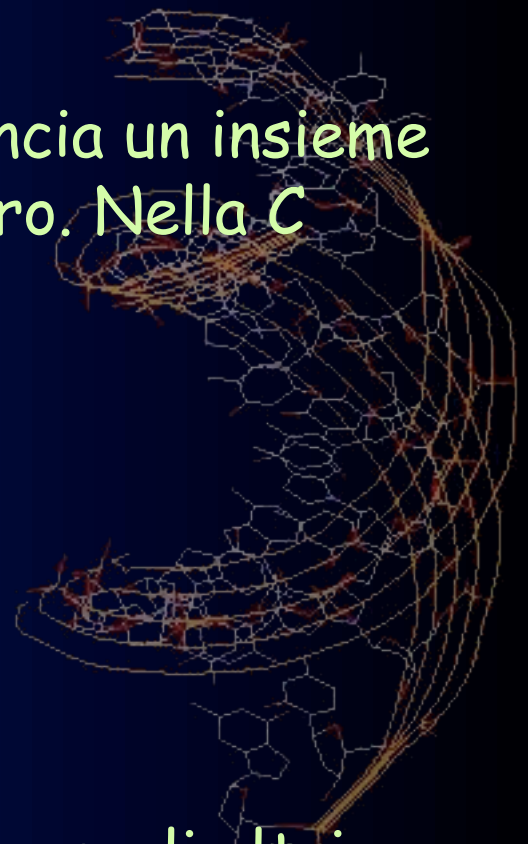
```
echo $shell
```

Visualizza la shell corrente

I file di startup

Quando si apre una shell, la shell stessa lancia un insieme di scripts che impostano l'ambiente di lavoro. Nella C shell questi scripts sono I seguenti:

- `/etc/csh.login`
- `$HOME/.cshrc`
- `$HOME/.login`



Il primo script e' gestito dal system manager, gli altri due sono modificabili dall'utente.


I costrutti del C

Il costrutto switch

Il costrutto switch è molto flessibile in quanto consente di eseguire differenti istruzioni in base ad un'espressione che può assumere più di due valori.

```
switch (expression)
{
    case  template_1: statement;
    case  template_2: statement;
    . . .
    case  template_n: statement;
    default: statement;
}
```

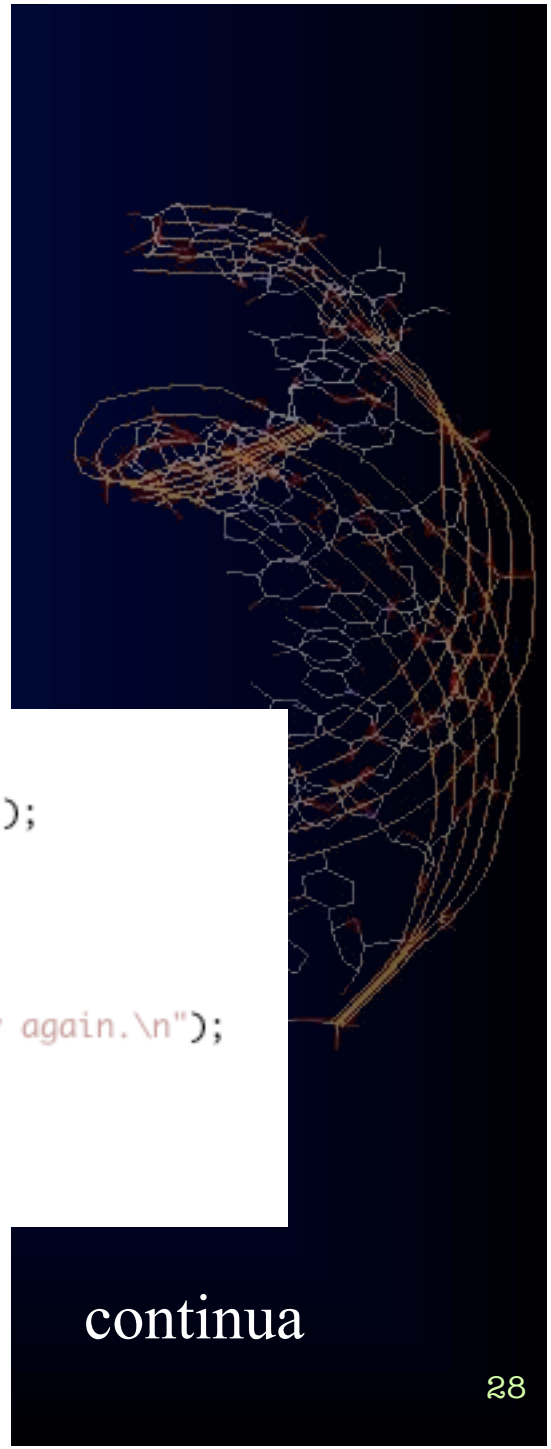
```c
/* Demonstrates the switch statement. */

#include <stdio.h>

int main()
{
    int reply;

    puts("Enter a number between 1 and 5:");
    scanf("%d", &reply);

    switch (reply)
    {
        case 1:
            puts("You entered 1.");
        case 2:
            puts("You entered 2.");
        case 3:
            puts("You entered 3.");
        case 4:
            puts("You entered 4.");
        case 5:
            puts("You entered 5.");
        default:
            puts("Out of range, try again.");
    }

    return 0;
}
```

INPUT/OUTPUT:

```
Enter a number between 1 and 5:
2
You entered 2.
You entered 3.
You entered 4.
You entered 5.
Out of range, try again.
```

```c
/* Demonstrates the switch statement correctly. */

#include <stdio.h>

int main()
{
int reply;

puts("\nEnter a number between 1 and 5:");
 scanf("%d", &reply);

 switch (reply)
 {
 case 0:
 break;
 case 1:
 {
 puts("You entered 1.\n");
 break;
 }
 case 2:
 {
 puts("You entered 2.\n");
 break;
 }
 case 3:
 {
 puts("You entered 3.\n");
 break;
 }
 case 4:
 {
 puts("You entered 4.\n");
 break;
 }
```

```c
case 5:
{
puts("You entered 5.\n");
break;
}
default:
{
puts("Out of range, try again.\n");
}
} /* End of switch */
return 0;
}
```

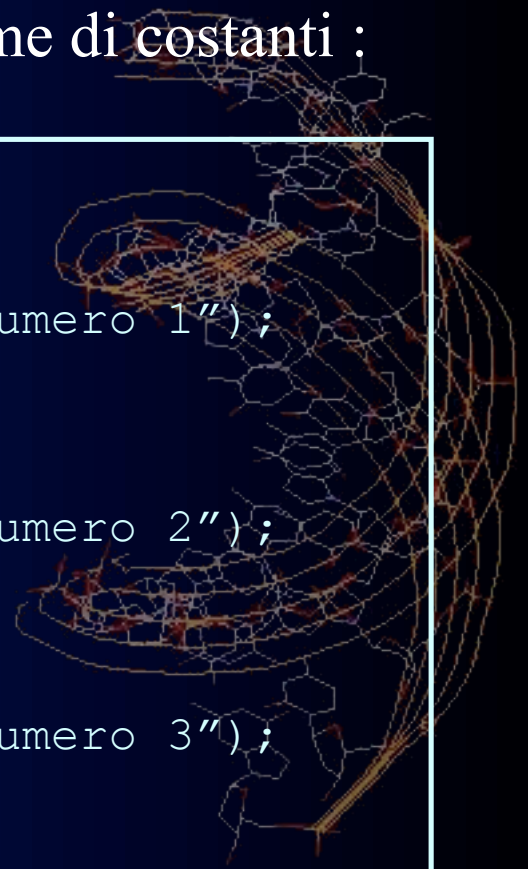


continua

Il costrutto switch

Viene usato per una scelta multipla fra un insieme di costanti :

```
switch (nBusta){
        case 1:
                printf("Hai scelto la busta numero 1");
                break;
        case 2:
                printf("Hai scelto la busta numero 2");
                break;
        case 3:
                printf("Hai scelto la busta numero 3");
                break;
        default:
                printf("Non hai scelto una busta valida");
                break;
}
```



Il costrutto switch

```
scanf("%c",&carattere_letto);
switch (carattere_letto){
      case 'a': case 'e' : case 'i' : case 'o':
      case 'u':
            printf("Hai scritto una vocale");
            break;
      default:
            printf("Hai scritto una consonante");
            break;
}
```

(cosa succede se inserisco il carattere '!' o il carattere 'A' ?)

DO & DON'T

Utilizzare sempre il caso di `default`

Utilizzare sempre il costrutto switch quando più di due alternative sono prese in considerazione per la stessa variabile

Allineare le istruzioni `case` per migliorare la leggibilità

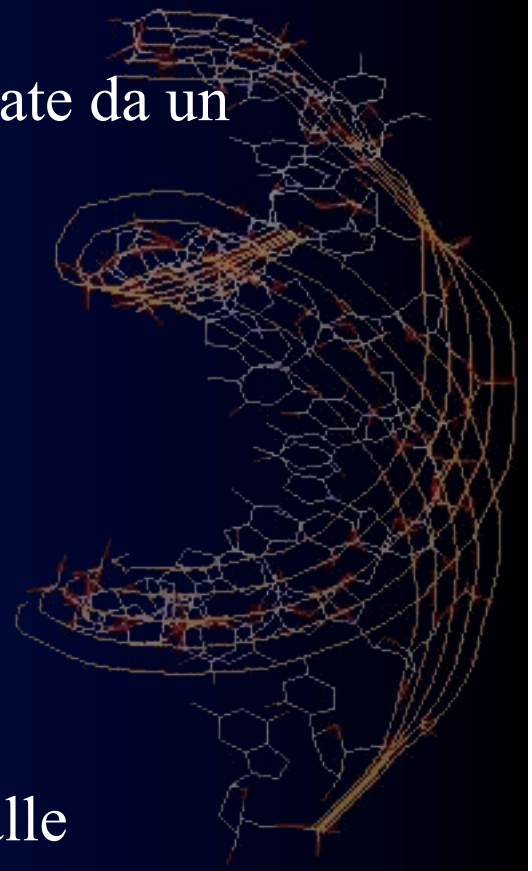
Non dimenticare di inserire l'istruzione `break` quando è necessaria

Strutture iterative

Array : insieme di locazioni di memoria identificate da un indice che contengono dati tra loro *omogenei*

Come le altre variabili gli arrays devono essere *dichiarati* prima del loro utilizzo

```
int data[1000];
```



DON'T

Non definire la dimensione degli arrays superiore alle reali esigenze

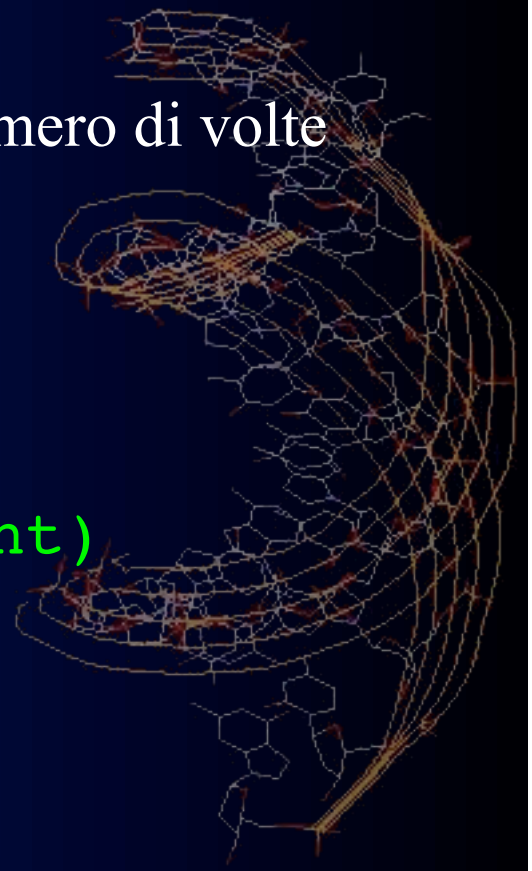
Non dimenticare che il primo elemento di un array corrisponde all'indice 0

Ciclo for

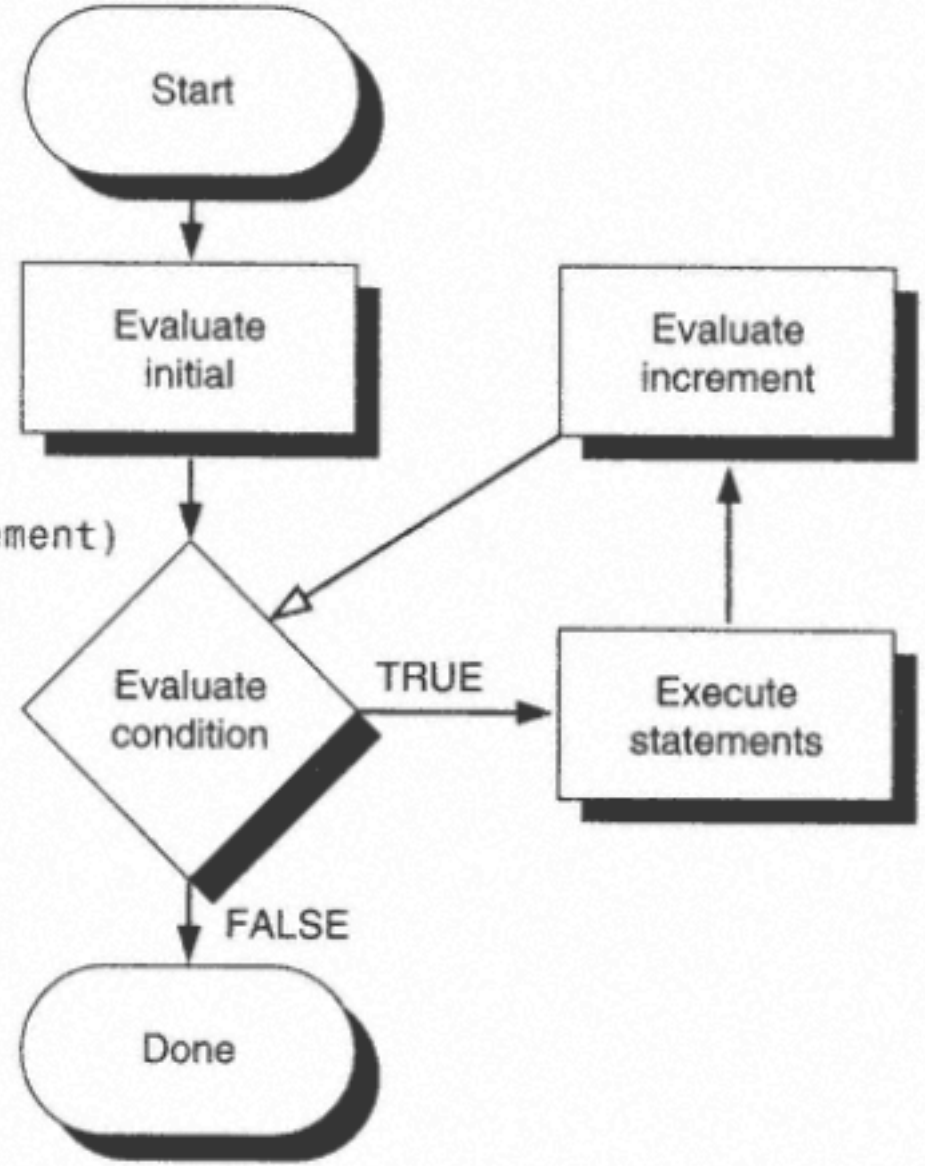
L'istruzione `for` consente di eseguire un certo numero di volte un blocco di una o più istruzioni

L'istruzione `for` ha la seguente struttura

```
for (initial; condition; increment)
     statement;
```

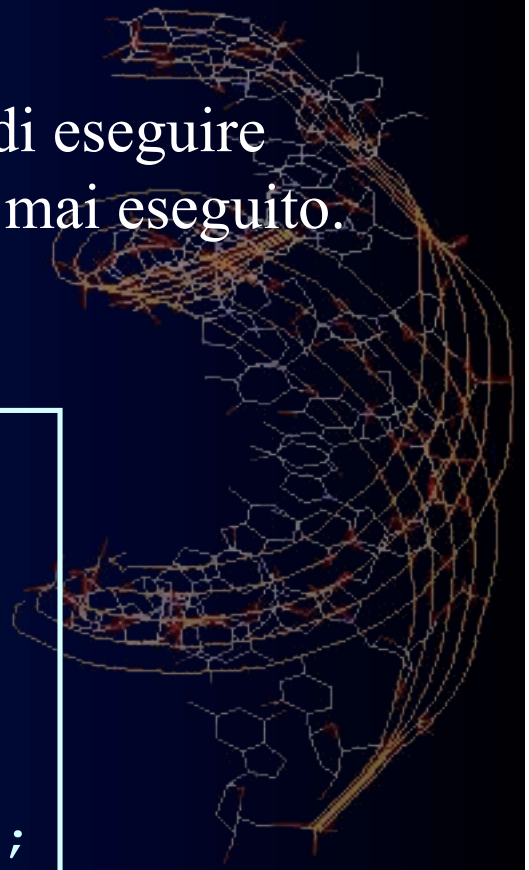


```
for (initial; condition; increment)
    statements
```



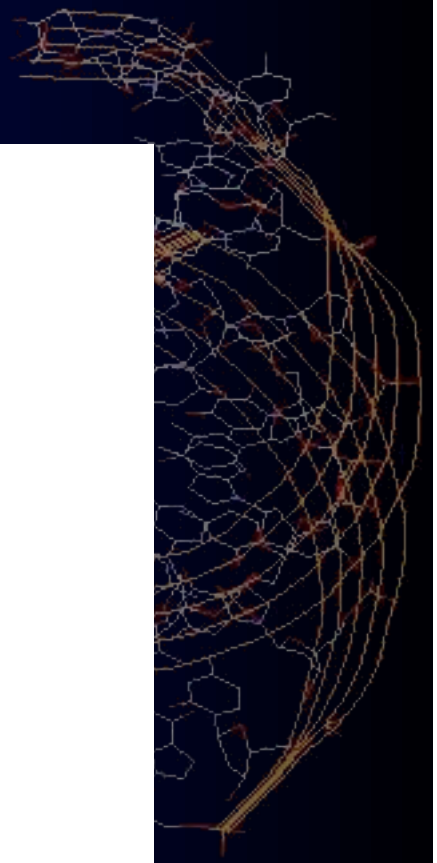
Ciclo for

La condizione di controllo viene verificata prima di eseguire qualunque istruzione: il ciclo potrebbe non venire mai eseguito.



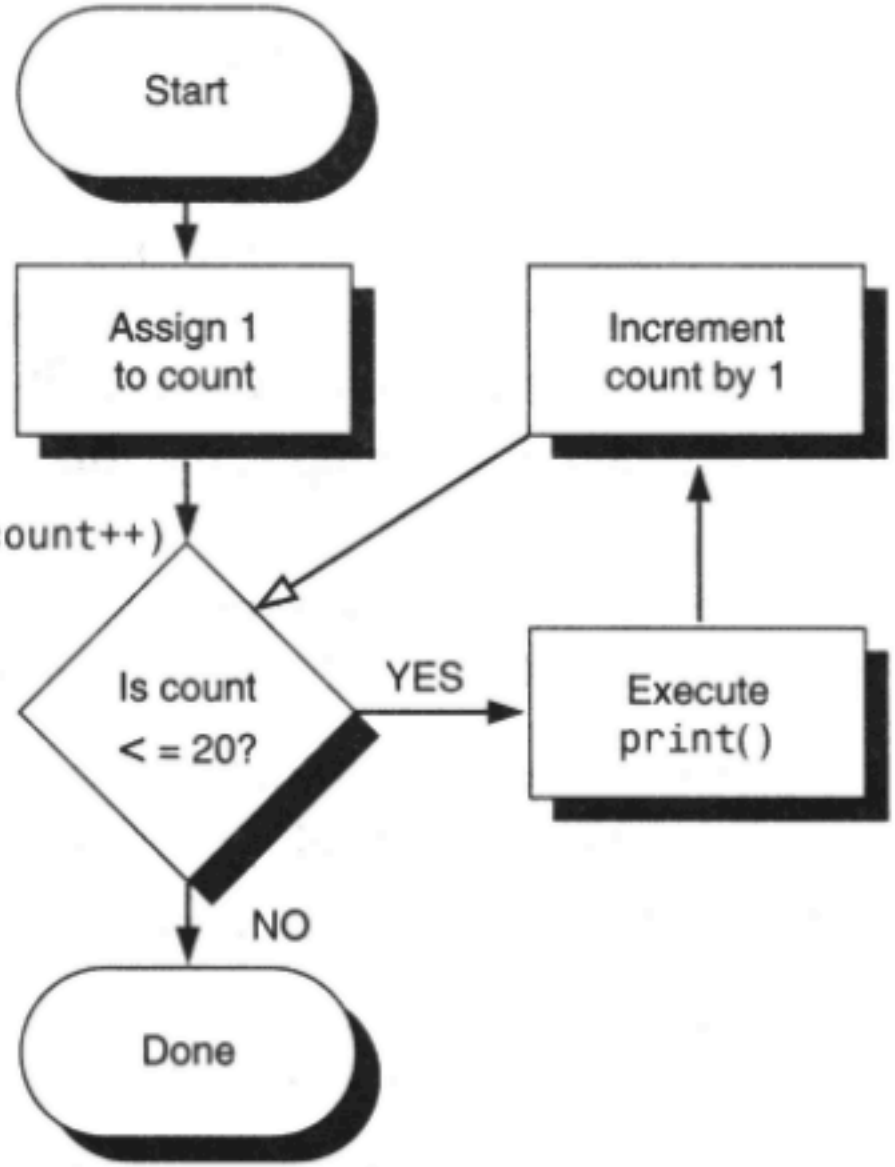
```
somma = 0;
for (i=0; i< 100; i++)
{
somma = somma + i;
}
printf("Somma = %d",somma);
```

Esempio



```
/* Demonstrates a simple for statement */

#include <stdio.h>

int count;

int main()
{
    /* Print the numbers 1 through 20 */

    for (count = 1; count <= 20; count++)
        printf("%d\n", count);

    return 0;
}
```

```
for (count = 1; count < = 20; count++)
     print ( "\n%d", count);
```

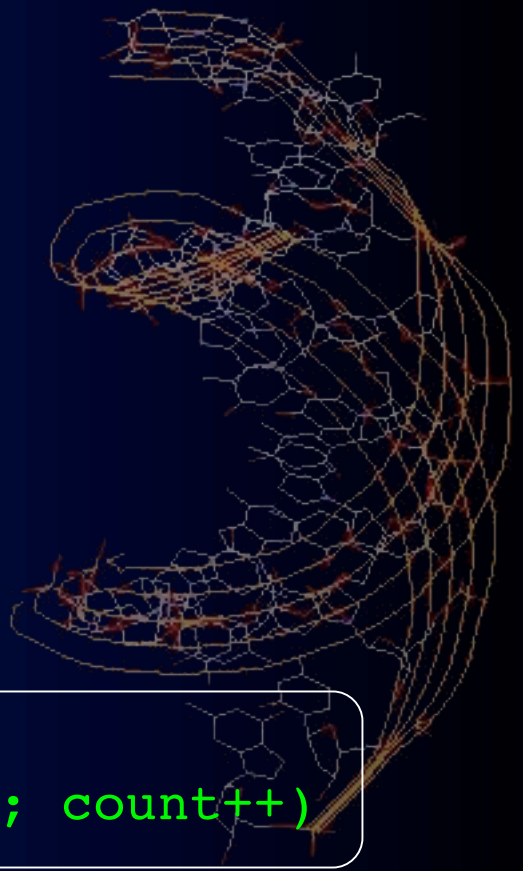


Ciclo `for`

L'istruzione `for` è molto flessibile.

Si può omettere, ad esempio, l'espressione di inizializzazione

```
count = 1;
for ( ; count < 1000; count++)
```



oppure

```
count = 1;
for (printf("Riordinamento") ; count < 1000; count++)
```

È anche possibile omettere l'istruzione di incremento

```
for ( count = 0; count < 1000; )
      printf("%d", count++);
```

Ciclo `for`

L'espressione di test che controlla l'esecuzione del ciclo può essere una qualsiasi espressione in C

```
for (count =0; count < 100 && array[count] != 0; count++)
      printf("%d", array[count]);
```

Che equivale a

```
for (count =0; count < 100 && array[count]; )
      printf("%d", array[count++]);
```

Ciclo `for`

L'utilizzo della virgola all'interno delle espressioni di un'istruzione `for` può essere estremamente utile.

Supponiamo, ad esempio, di avere due arrays di 100 elementi (`a[ ]` e `b[ ]`) e di voler copiare il contenuto di a[ ] in b[ ] in ordine inverso

`b[0] = a[99], b[1] = a[98], …`

```
for (i = 0, j = 99; i < 100; i++, j--)
      b[j] = a[i];
```

Example 1

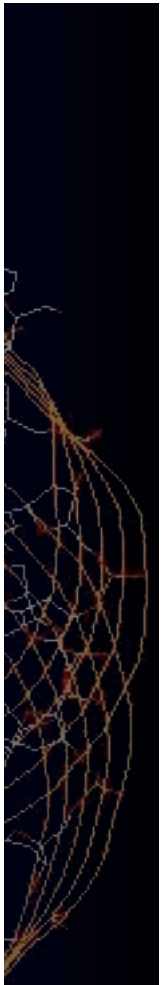
```
int x;
for (x = 0; x <10; x++)
    printf( "\nThe value of x is %d", x );
```

Example 2

```
int nbr = 0;
for ( ; nbr != 99; )
   scanf( "%d", &nbr );
```

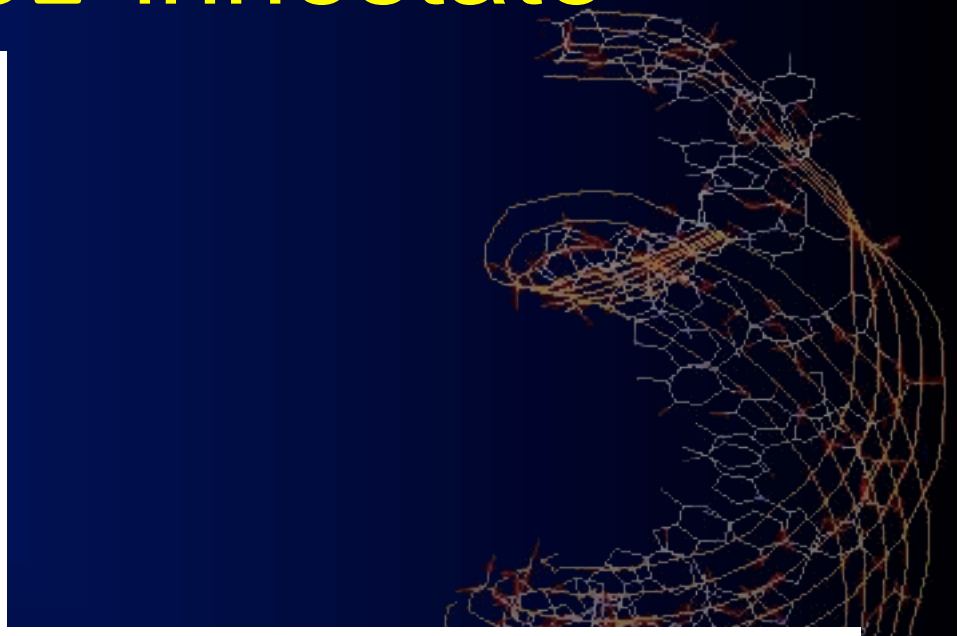
Example 3

```
int value[10];
int ctr,nbr=0;
for (ctr = 0; ctr < 10 && nbr != 99; ctr++)
{
    puts("Enter a number, 99 to quit ");
    scanf("%d", &nbr);
    value[ctr] = nbr;
}
```



Istruzioni `for` innestate

```c
/* Demonstrates nesting two for statements */

#include <stdio.h>
void draw_box( int, int);

int main()
{
draw_box( 8, 35 );

 return 0;
 }

 void draw_box( int row, int column )
 {
 int col;
 for ( ; row > 0; row--)
 {
 for (col = column; col > 0; col--)
 printf("X");

 printf("\n");
 }
 }
```

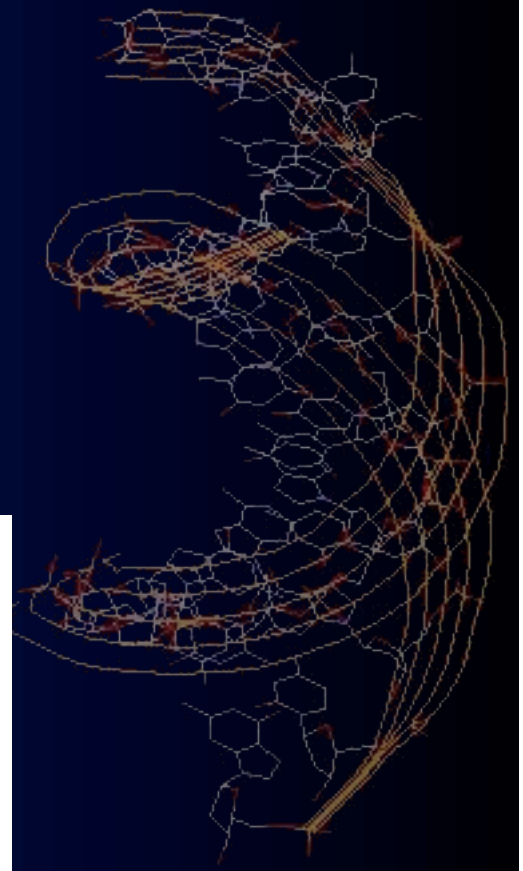


```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

BUG BUSTER

```
switch( answer )
{
    case 'Y': printf("You answered yes");
              break;
    case 'N': printf( "You answered no");
}
```

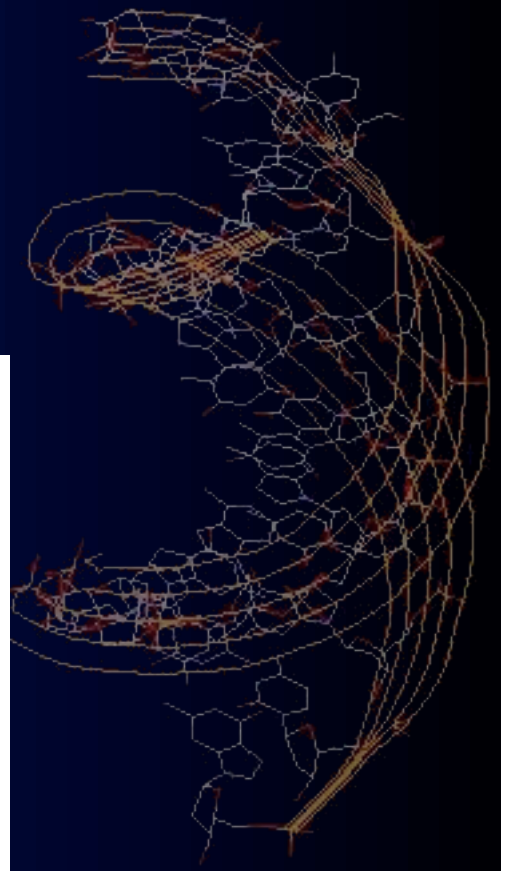
```
switch( choice )
{
    default:
        printf("You did not choose 1 or 2");
    case 1:
        printf("You answered 1");
        break;
    case 2:
        printf( "You answered 2");
        break;
}
```



Homework

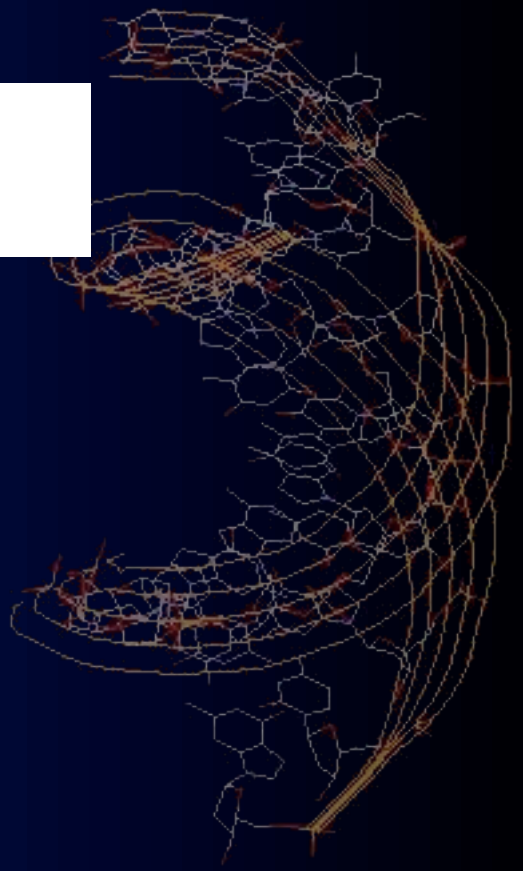
Riscrivere le seguenti linee di codice usando il costrutto `if`

```
switch( choice )
{
    default:
        printf("You did not choose 1 or 2");
    case 1:
        printf("You answered 1");
        break;
    case 2:
        printf( "You answered 2");
        break;
}
```



BUG BUSTER

```
for (counter = 1; counter < MAXVALUES; counter++);
    printf("\nCounter = %d", counter );
```



Homework

Quanto vale **x** al termine del seguente ciclo ?

```
for (x = 0; x < 100, x++) ;
```

Quanto vale **ctr** al termine del seguente ciclo ?

```
for (ctr = 2; ctr < 10; ctr += 3) ;
```

Scrivere un ciclo **for** per contare da 1 a 100 di 3 in 3