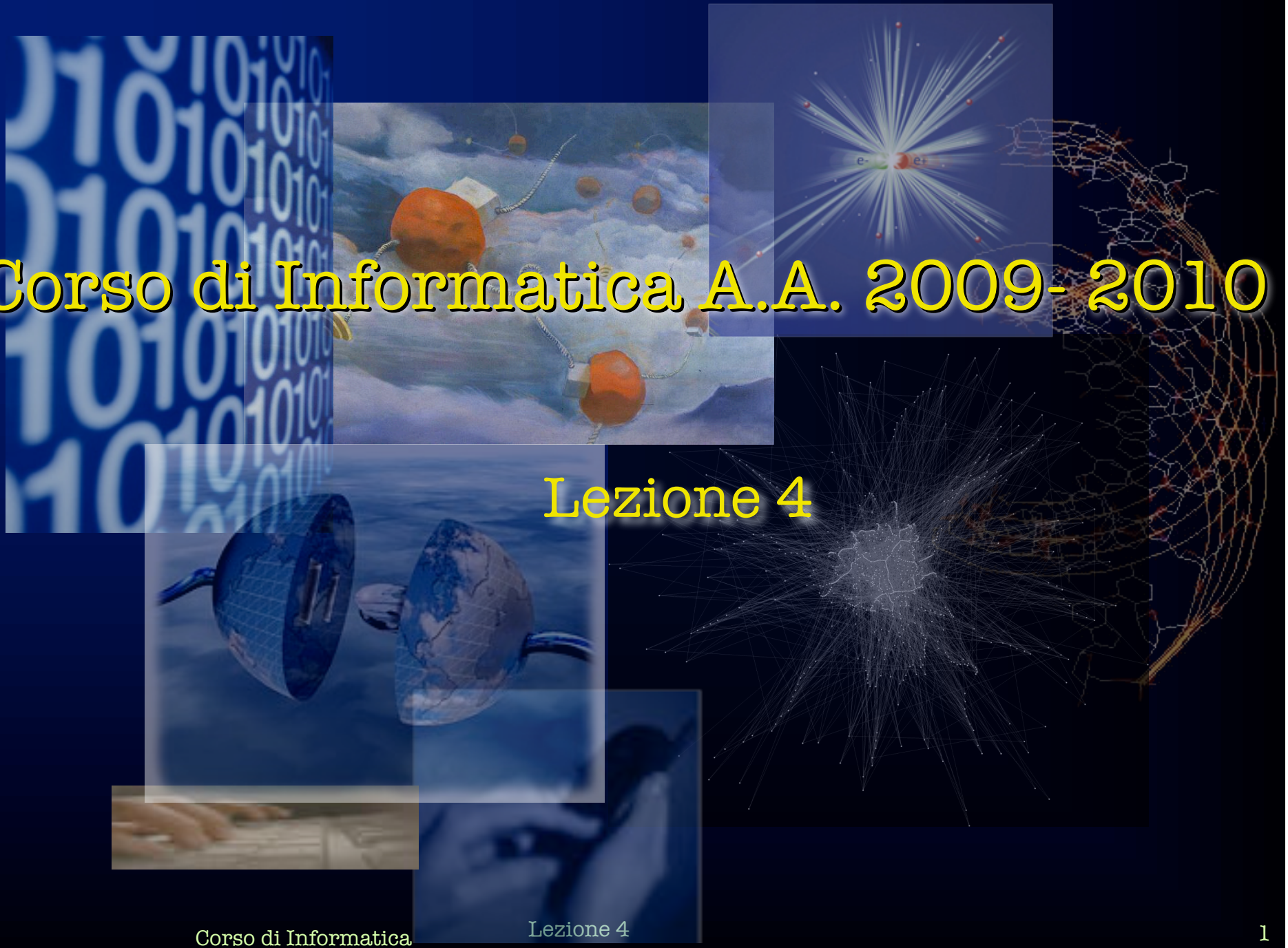


Corso di Informatica A.A. 2009-2010

Lezione 4

Funzioni del Sistema Operativo

- Concetto di OS
  - Gestione dell'I/O
  - Gestione del file system

Gestione dell' I/O

L'OS fornisce dei devices virtuali, che permettono ad esempio al programmatore di prevedere un'istruzione di tipo

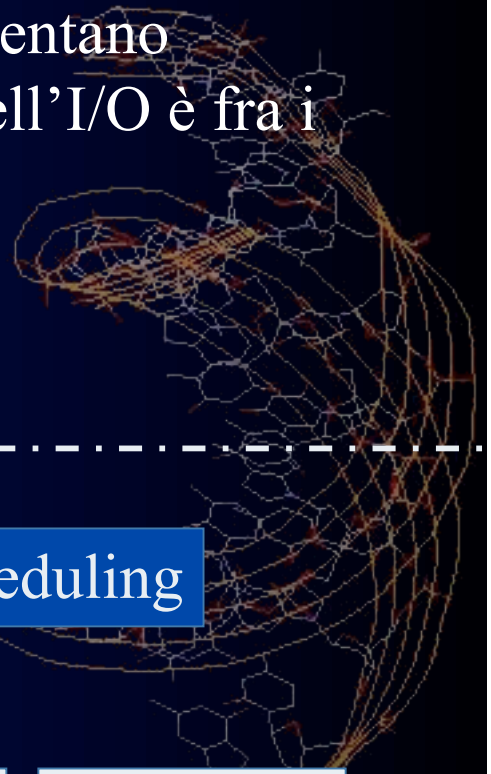
```
printf("Hello world \n");
```

per stampare a video la frase Hello world, senza preoccuparsi se il monitor è un Philips 17" o un Sony e se è collegato su una scheda PCI o AGP…

A livello hardware l'OS poi gestisce fisicamente i devices tramite i device drivers. Uno dei compiti in assoluto più complessi e delicati è la gestione degli interrupts e l'assegnazione delle risorse, per evitare di incorrere in circoli viziosi (deadlocks).

Gestione dell'I/O

I dispositivi di I/O sono di tipo molto vario, e presentano ciascuno peculiari caratteristiche => la gestione dell'I/O è fra i compiti più complessi di un OS.



Livello utente

Interfaccia I/O dell'applicazione:
Virtual device

Livello kernel (privilegiato)

Funzioni comuni di I/O : buffering e scheduling

Driver tastiera | Driver mouse | Driver Hard disk | Driver video | Driver audio

Hardware: controllers di tastiera, mouse, disco, video, audio etc.

File & Directories

- File: contenitore logico di informazione immagazzinato nella memoria di massa. Può contenere programmi, testi, immagini, suoni, etc..

- Directory o cartella: area della memoria di massa che contiene files o altre cartelle (sottocartelle). Di fatto e' un file contenente l'elenco dei files appartenenti a quel gruppo e le infomazioni che permettono al s.o. di localizzare i files.

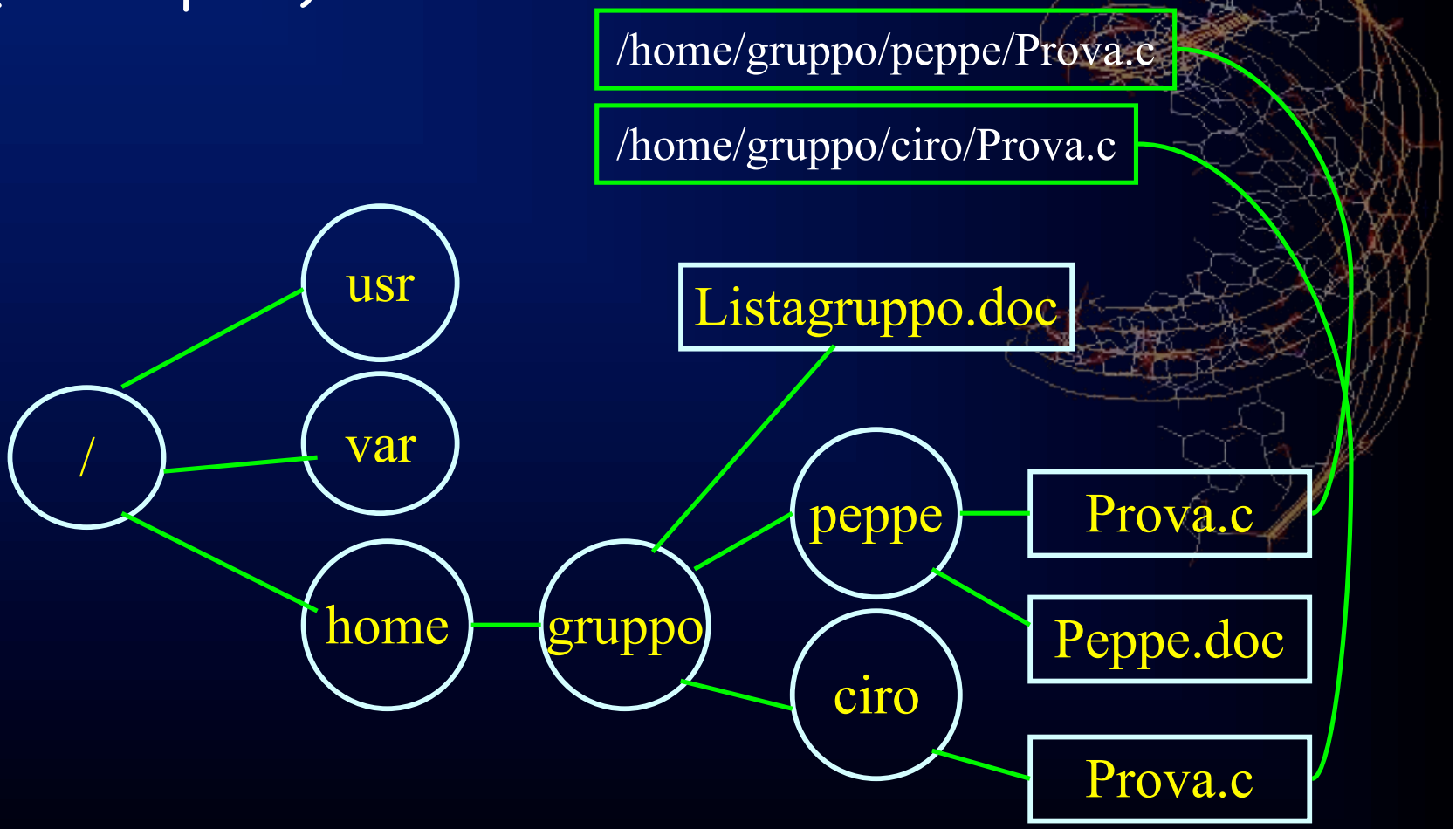
Il file system

I servizi che un OS deve fornire per la gestione dei files sono, essenzialmente:

1. Fornire un metodo per associare un nome ad un file
2. Manipolare l'accesso, la creazione e la cancellazione dei files
3. Gestire la corrispondenza fra blocchi logici e blocchi fisici in modo trasparente all'utente
4. Realizzare meccanismi di protezione dei dati per determinare chi può avere accesso o controllo sui vari files.

Directories e organizzazione gerarchica

Una directory è un particolare file, contenitore di files e directory. Essa definisce l'ambito di validità di un nome file (name space).



Controllo di Accesso

Il creatore di un file dovrebbe essere in grado di controllare ciò che può essere fatto con quel file e *da chi*.

Sono possibili varie modalità di accesso, che possono essere permesse a singoli utenti o a gruppi di utenti:

- Read
- Write
- Execute
- Append
- Delete
- List
- ...

Linguaggi ad alto e basso livello

Come visto la macchina di Von Neumann è capace di eseguire sequenze ben ordinate di istruzioni in linguaggio macchina del tipo:

```
0110100101000100-0010011000001101-0001…
```

```
Codice istruzione-indirizzo1…indirizzoN
```

L'istruzione deve essere presente nel set di istruzioni della ALU.

Un programma in L.M. è intrinsecamente legato all'hardware per cui è stato realizzato. Il L.M. è un linguaggio di *basso livello*.

I linguaggi che permettono di svincolarsi dalla rappresentazione hardware vengono chiamati linguaggi di *alto livello.*

Assembler

Il primissimo passo nella gerarchia dei linguaggi è il linguaggio assembler. Si tratta di un linguaggio di basso livello che però ammette una rappresentazione "human readable" del programma.

Linguaggio Macchina
01000101010100100 1
00101010100010100 1
11001010100100100
11100101001010010 0

Linguaggio Assembler
LOAD R1,X
LOAD R2,Y
ADD R3,R1,R2
STORE R3,Y

Il linguaggio assembler è una semplice traduzione del L.M., i comandi assembler ammettono una trascrizione 1 a 1 con quelli della ALU, e sono quindi specifici per ogni CPU.

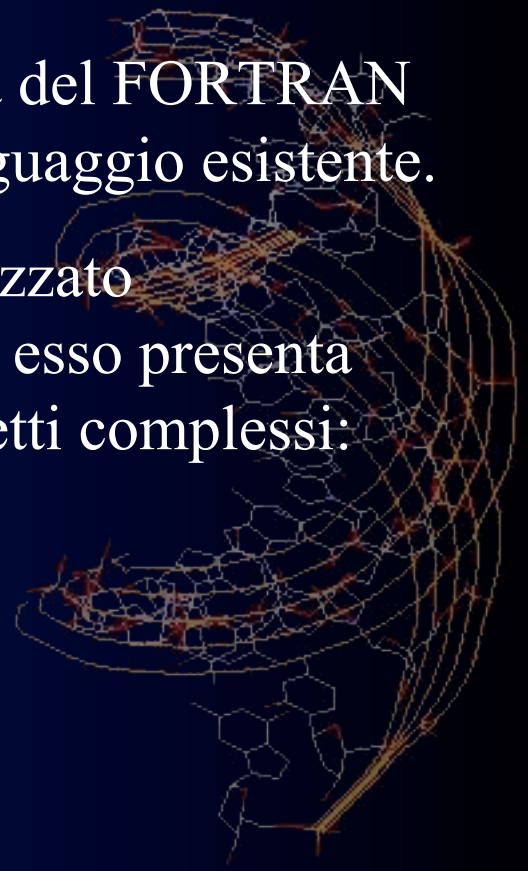
Un *assemblatore* è quella parte del sistema operativo che trascrive un programma in assembler nel suo corrispondente in L.M.

Linguaggi ad alto livello (I)

Agli albori della programmazione (fino alla nascita del FORTRAN alla fine degli anni ’50) l’assembler era l’unico linguaggio esistente.

Sebbene l’assembler possa in principio essere utilizzato direttamente per implementare qualsiasi algoritmo esso presenta numerose gravi limitazioni per un utilizzo in progetti complessi:

- Portabilità
- Leggibilità
- Riutilizzabilità
- Controllo degli errori



Questo ha portato alla creazione di linguaggi più vicini al linguaggio naturale e più lontani da quello della macchina….

Linguaggi ad alto livello (II)

Un linguaggio di alto livello NON ammette una semplice traduzione 1 a 1 con le istruzioni della macchina, ma definisce costrutti e comandi specifici del linguaggio e indipendenti dalla macchina su cui devono essere eseguiti.

Il processo di traduzione in L.M. (l'unico veramente compreso dal calcolatore) è quindi di gran lunga più complesso che per un programma assembler.

Dall'algoritmo al processo (I)

Algoritmo/Pseudocodice

Editor o ambiente di sviluppo grafico

Programma Sorgente

Preprocessore

COMPILATORE

Analisi lessicale : "ortografia"

Analisi sintattica (parsing) : "grammatica/logica"

Analisi semantica: "significato"

Ottimizzazione, assembling

File oggetto

Dall'algoritmo al processo (II)

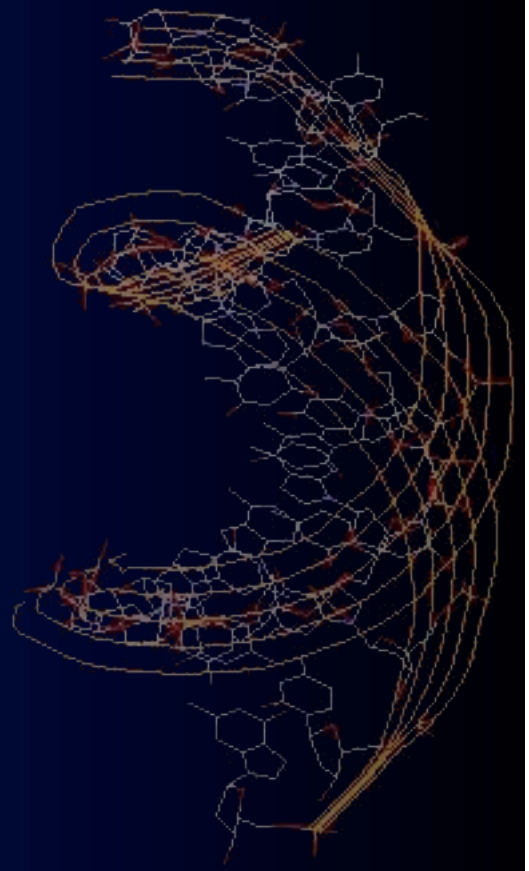
File oggetto

Linker + librerie esterne

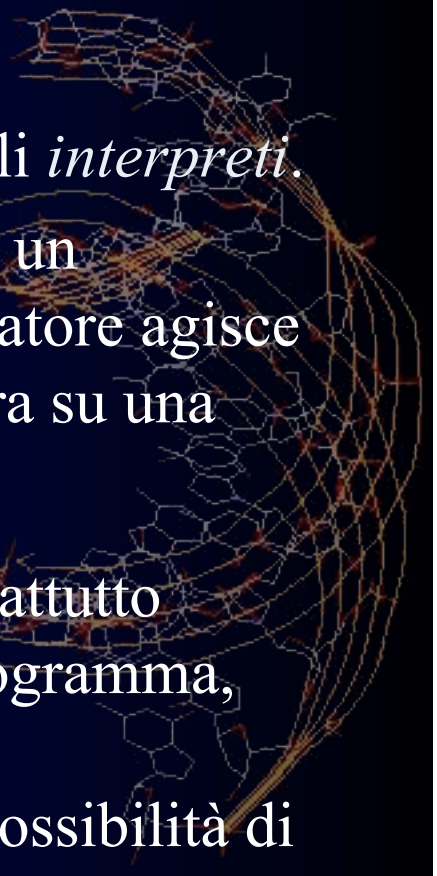
Programma eseguibile

Loader

Processo in memoria



Interpreti



Oltre ai compilatori un altro tipo di traduttori sono gli *interpreti*.

Per molti versi l'interprete svolge un ruolo simile ad un compilatore, con la differenza che mentre un compilatore agisce globalmente su tutto il programma, l'interprete lavora su una istruzione alla volta, traducendola ed eseguendola.

Lo svantaggio di utilizzare un interprete risiede soprattutto nell'impossibilità di ottimizzare l'esecuzione del programma, mentre i suoi vantaggi sono soprattutto la semplice localizzazione degli errori di programmazione e la possibilità di utilizzare direttamente il codice sorgente su macchine diverse senza dover effettuare nuovamente la compilazione del codice.

Programmi di utilità

Oltre agli elementi essenziali già menzionati il sistema operativo può fornire tutta una serie di servizi aggiuntivi di utilità, pensati per le applicazioni più comuni di un calcolatore. Tra questi citiamo ad esempio:

- Word processing
- Navigazione Web e gestione e-mail
- Fogli elettronici
- Manipolazione files multimediali
- Strumenti di gestione di database
- …

Utilizzo del sistema operativo

- Interfaccia a linea di comando e GUI
- Navigazione nel file system
- Operazioni di Base in una GUI

Interfaccia a linea di comando

I primi sistemi operativi (OS) erano del tipo a linea di comando: l'utente doveva immettere attraverso la tastiera il comando voluto e il sistema lo eseguiva.

L'interfaccia a riga di comando è ancora presente nella gran parte dei OS a fianco delle moderne GUI (Graphical User Interfaces)

Vantaggi:

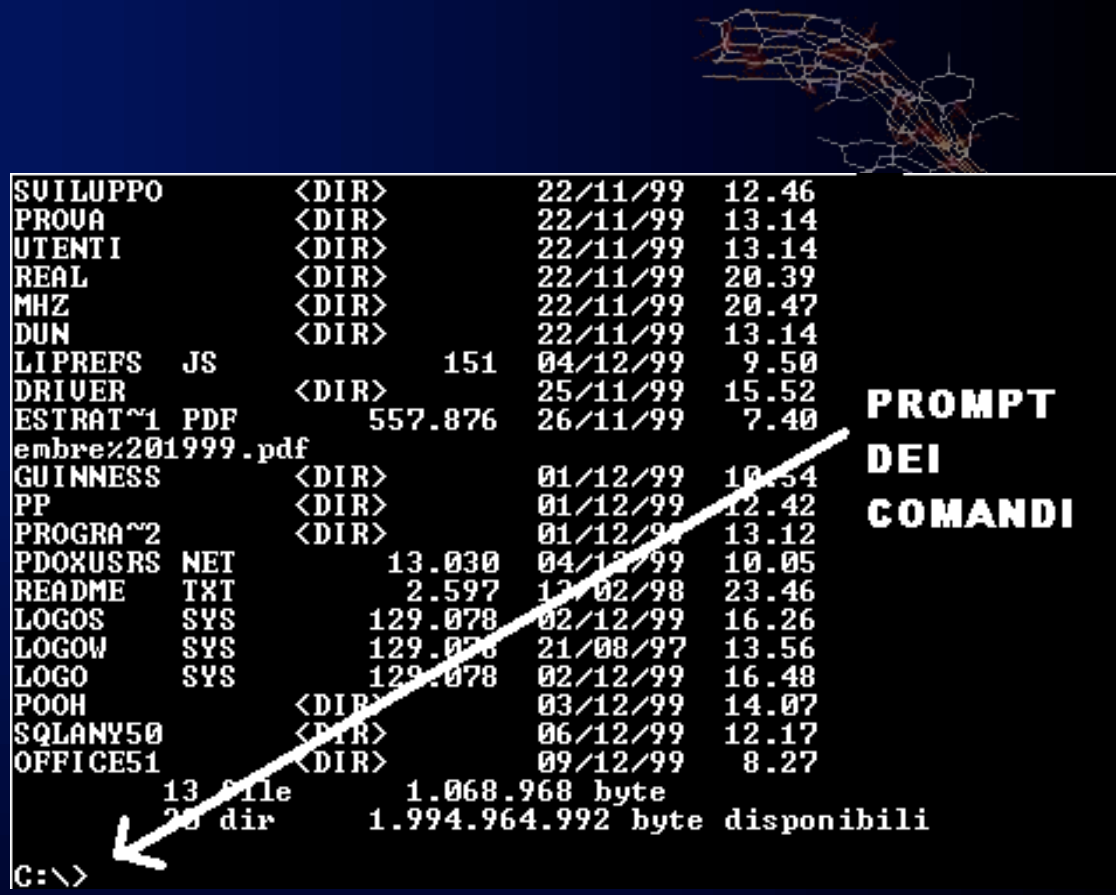
- Operazioni complesse possono essere eseguite più semplicemente
- L'utente dopo la difficoltà iniziale acquisisce "naturalmente" una conoscenza più approfondita del calcolatore

Svantaggi:

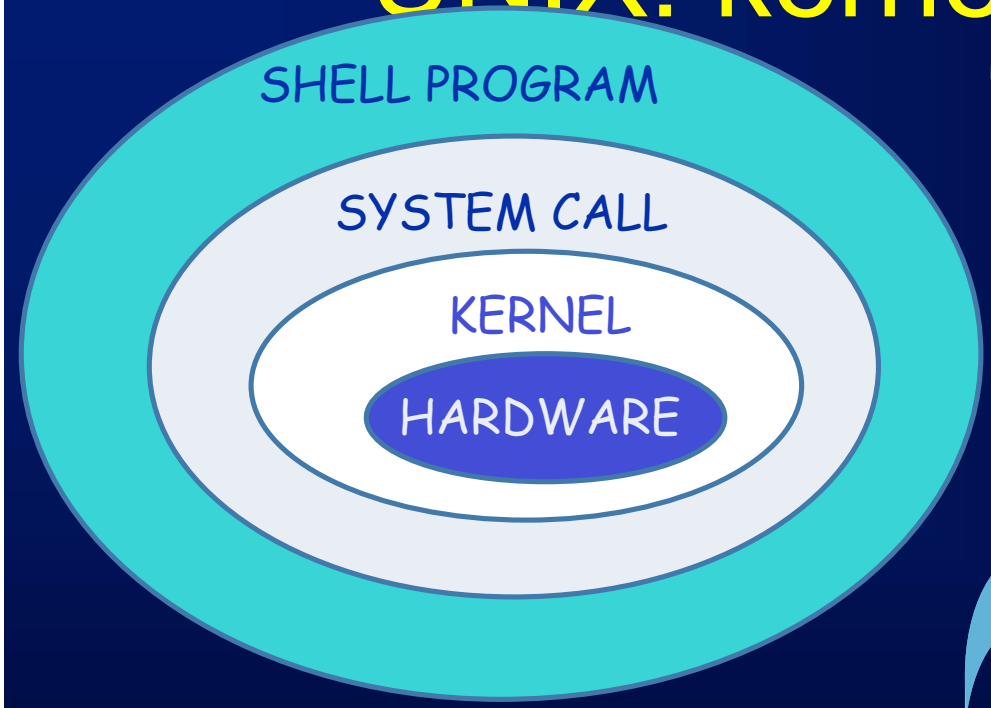
- Difficile l'interazione a prima vista

MS-DOS

- ✓ sviluppato nel 1981 da **Microsoft** per supportare il primo PC di **IBM**
- ✓ e' un sistema monoutente e monotask
- ✓ la Shell e' a linea di comando
- ✓ ha equipaggiato la quasi totalità dei PC prodotti fino al 1995
- ✓ e' disponibile solo per processori INTEL



UNIX: kernel & shell (I)



KERNEL: nucleo del s.o.

Il ***kernel*** gestisce

- la comunicazione con l'hardware "a basso livello",
- l'esecuzione dei programmi e
- l'uso e la condivisione o la protezione del sistema.

La ***shell*** e' un programma che interpreta i comandi dell'utente (programmi scritti con le system calls) e attiva i processi corrispondenti.

Le ***system call*** costituiscono lo strato successivo. Queste trattano l'hardware con elegante omogeneita'. Ogni dispositivo (disco, terminale, cdrom,…) e' trattato come se fosse un file.

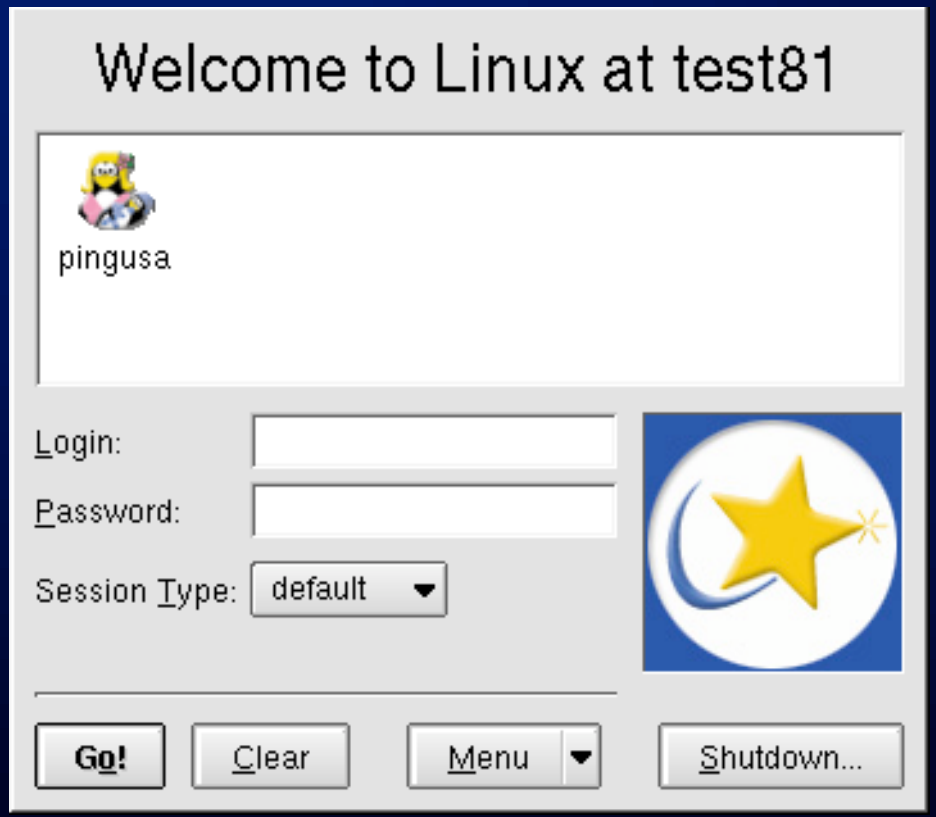
UNIX: kernel & shell (II)

La struttura a shell di UNIX ha due enormi vantaggi

- E' possibile modificare o sostituire la shell senza dover intervenire sul kernel, e questo ha portato ad una proliferazione delle shell (adattabilita' dell'ambiente di lavoro).
- La modifica dell'harware comporta solo l'aggiornamento del kernel (adattabilita' all'hardware)

Queste due caratteristiche rendono UNIX un s.o. aperto e facilmente trasportabile su hardware diversi, e ne giustificano la longevita' ed il suo largo impiego nel mondo scientifico.

….la prima volta… o quasi….



Login manager

Comandi (I)

Una volta collegati, il s.o. mette a disposizione dell'utente la possibilita' di inviare comandi.

La shell puo' acquisire i comandi in due forme:

1. con l'interazione diretta tra operatore e terminale (interprete di comandi);
2. leggendo file di testo (shell scripts) contenenti una serie di comandi da eseguire in sequenza (interprete di un programma).

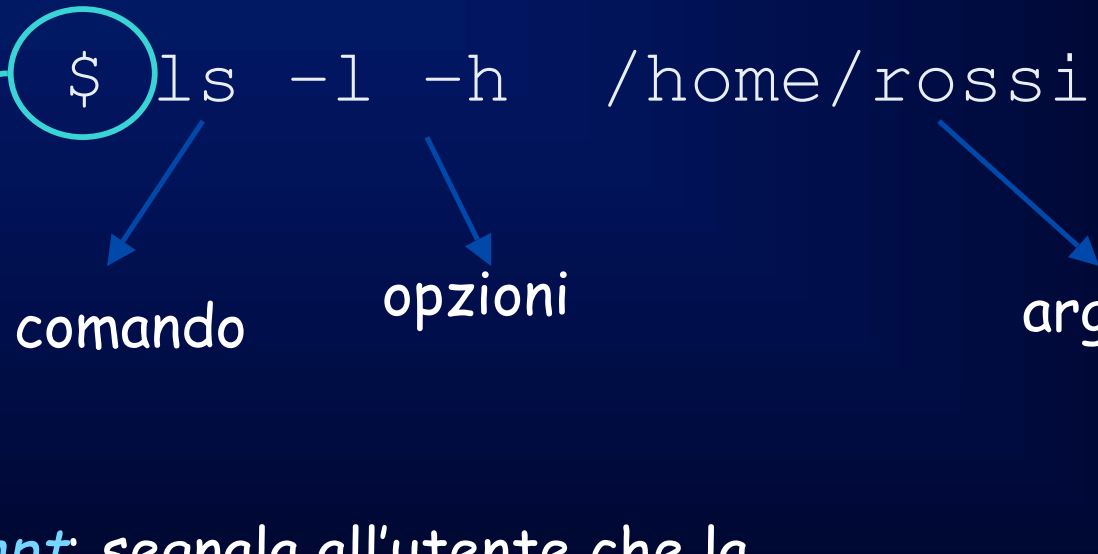
Comandi (II)

La shell di Unix mette a disposizione un grandissimo numero di comandi che possono anche essere considerati come dei filtri, nel senso che prendono dati in ingresso, eseguono su di loro un certo numero di operazioni e restituiscono dei dati in uscita.

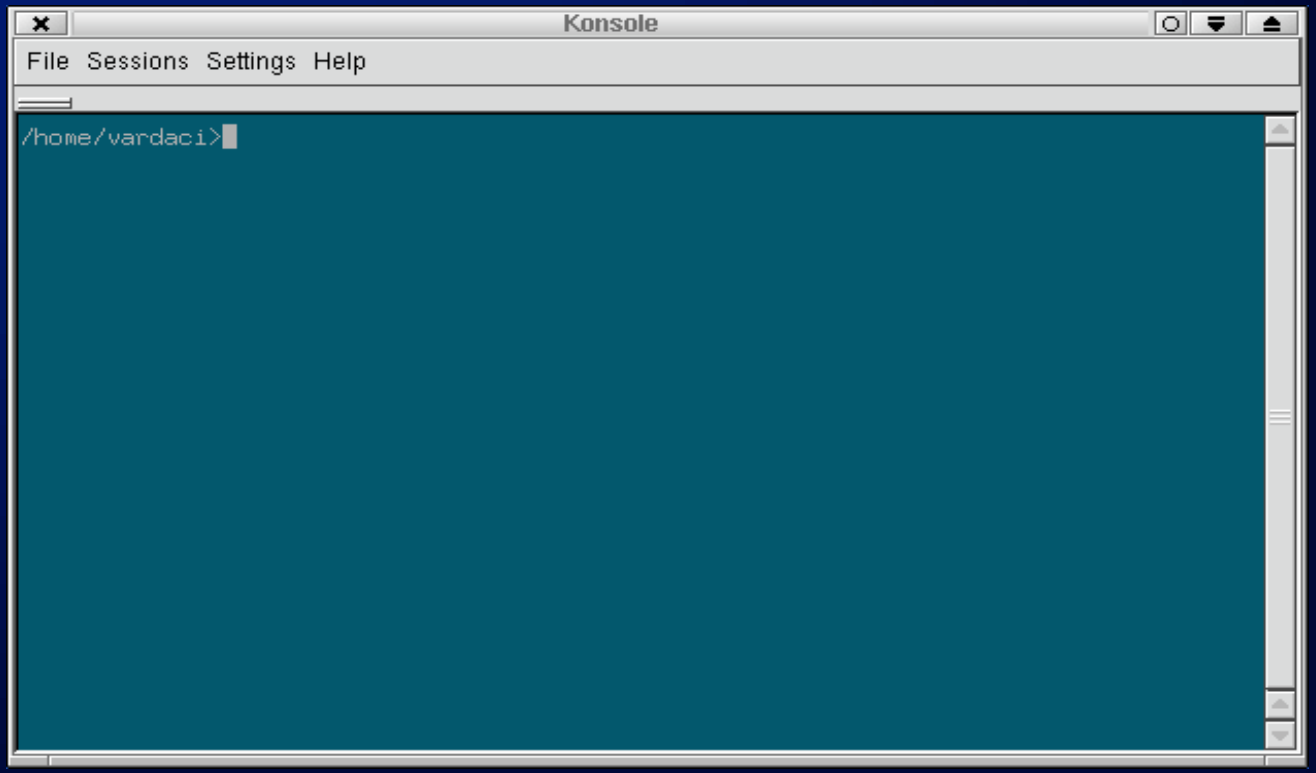
Nonostante questa grande varietà di comandi la struttura sintattica degli stessi risulta essere abbastanza semplice e standard, tanto che in linea di massima è possibile riassumere la forma sintattica come:

Comandi (III)

Es. Ottenere la lista dei file contenuti nella directory /home/rossi.



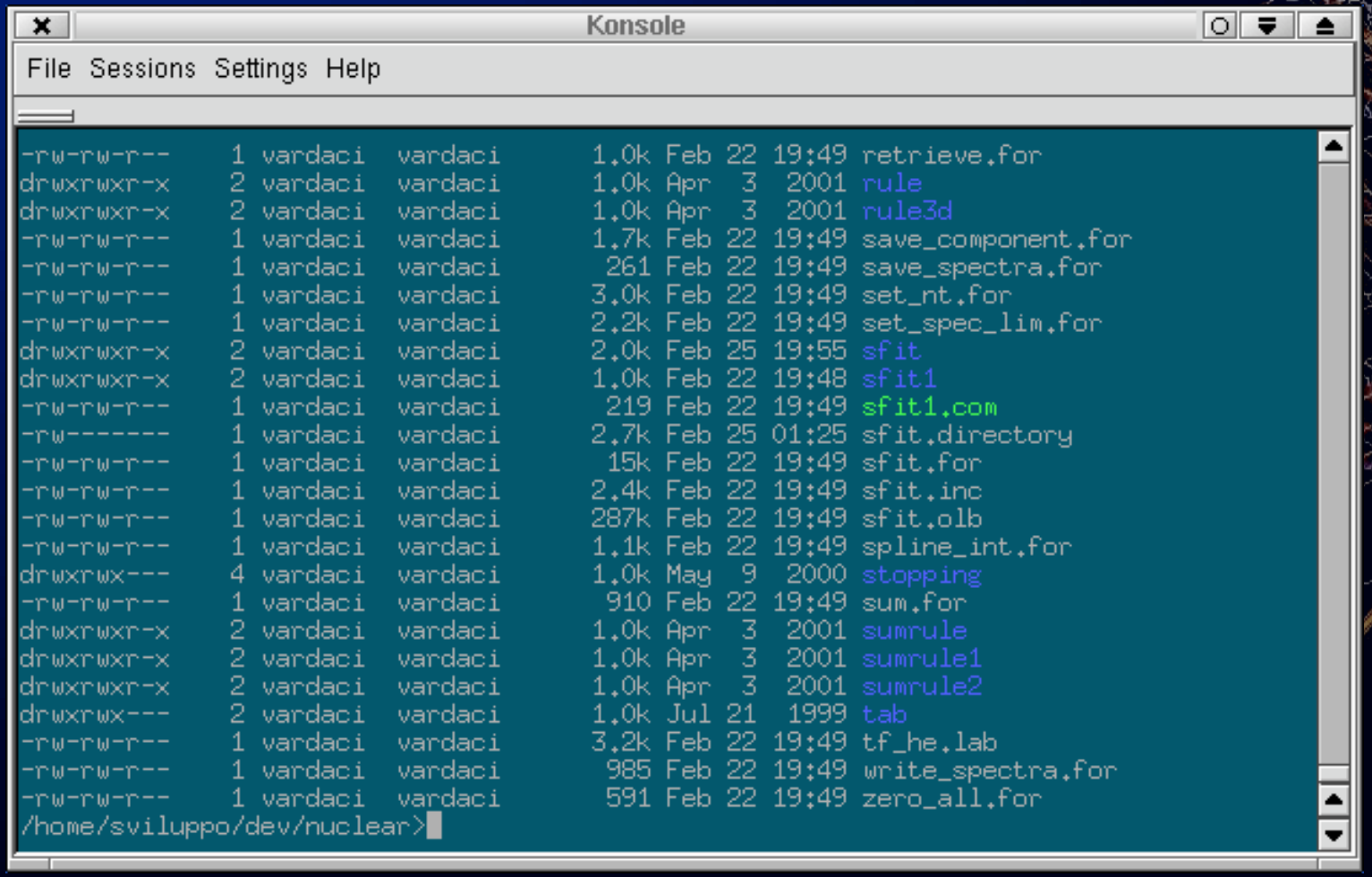
Interazione shell-terminale (I)



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.

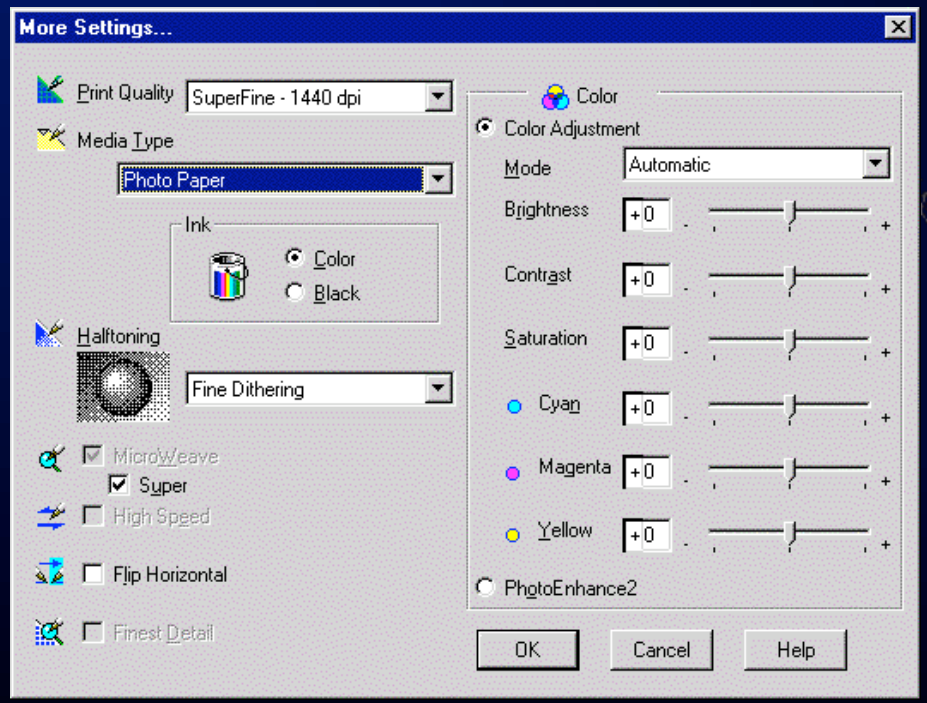
E' possibile navigare nel file system con il comando `cd`.

Interazione shell-terminale (II)



Graphical User Interfaces

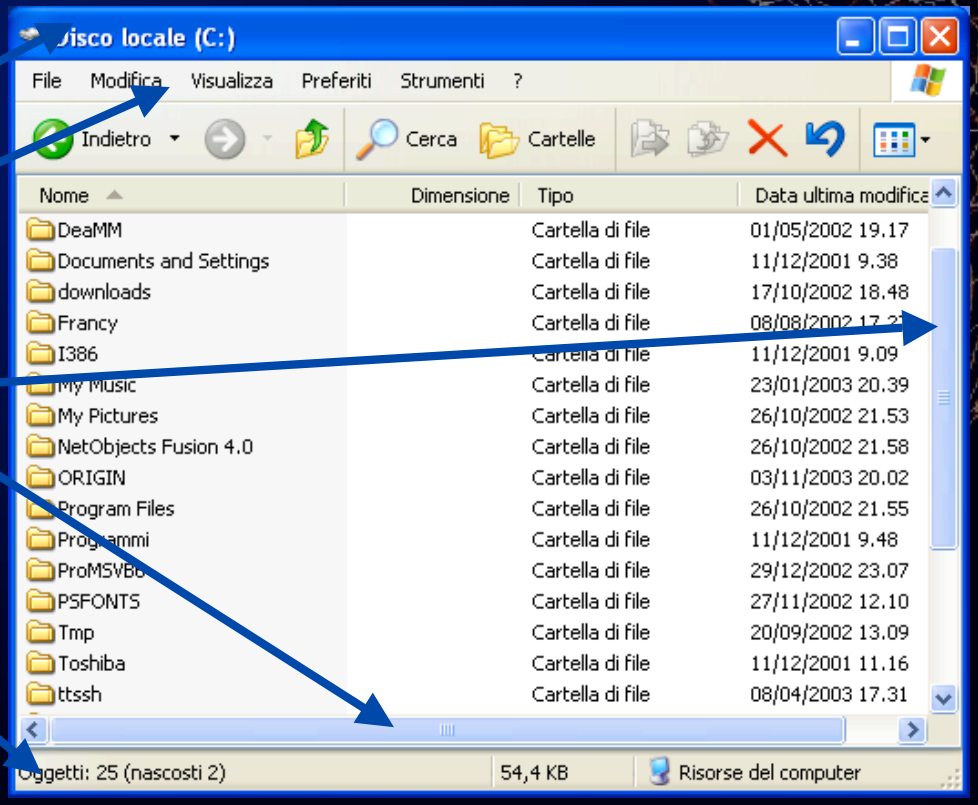
Le interfacce grafiche o GUI, nate con il sistema operativo Macintosh di Apple nel 1984 utilizzano delle metafore grafiche sottoforma di immagini (icone) e animazioni e l'utilizzo del mouse per effettuare le azioni.



Finestre e barre

Una finestra è una regione rettangolare con vari elementi grafici di controllo e una zona interna di lavoro. Le barre principali che contraddistinguono una finestra sono :

- Barra del titolo
- Barra dei menù
- Barre di scorrimento
- Barra di stato



Aprire un file

Che significa “Aprire” un file ? In generale significa accedere all’informazione che è contenuta nel file.

- Se il file è un semplice file ASCII un apposito programma (text editor) mi mostrerà la sequenza di caratteri che contiene.
- Se è un’immagine un programma di visualizzazione la mostrerà sullo schermo.
- Se è un audio un programma lo farà suonare dalle casse del computer etc.

In generale a seconda del tipo di file dovrò eseguire un tipo diverso di programma per visualizzarlo. Per facilitare il riconoscimento del tipo di file vengono comunemente utilizzate le ***estensioni*** ovvero gruppi di lettere (tipicamente tre) poste dopo un punto alla fine del nome del file stesso.

Estensioni comuni

.exe (es. explorer.exe) =	Files eseguibili ovvero programmi
.html (es. Mypage.html) =	Hypertext Markup Language ovvero pagine Web. Visualizzati da browser Web come Internet Explorer o Netscape
.txt (es. elenco.txt) =	Files ASCII (testo non formattato). Visualizzati da semplici text editor come Notepad (Blocco Note)
.doc (es. tesi.doc) =	Documenti di testo formattato di tipo di quelli creati da Microsoft Word . NON sono files ASCII…

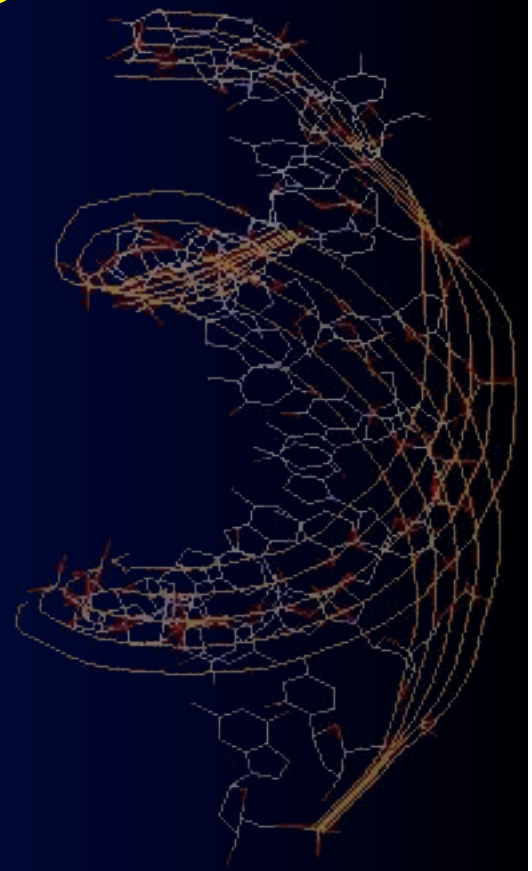
Estensioni comuni (II)

.bmp .tiff .jpg	Immagini. Apribili con programmi come ACDSee o iPhoto
.mp3	Files audio compressi. Apribili da programmi come WinAmp o Windows Media Player
.mpg .mpeg	Files video compressi. Apribili da programmi come Windows Media Player
.xls	•Fogli di lavoro nel formato del tipo di quelli creati da Microsoft Excel


Linguaggio C

Le Strutture del C

- Condizionali
  - `if-else` (`else if`)
  - `switch`
- Iterative
  - `while`
  - `for`
  - `do while`
  - `break` and `continue`



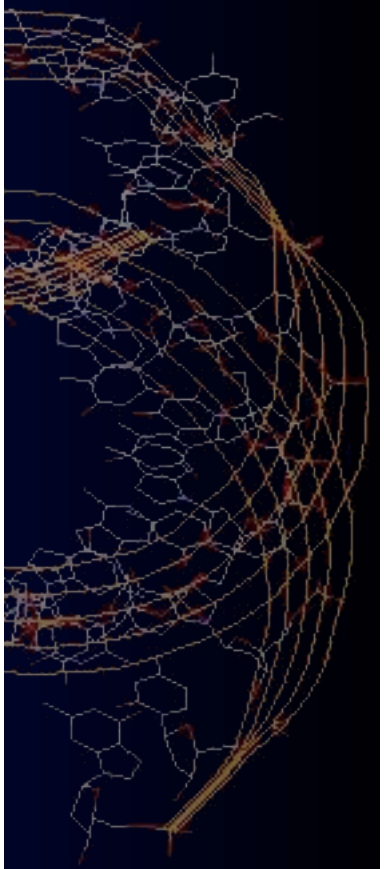
Il costrutto `if`

La forma generale di un'istrzuione `if` è la seguente

```
if (expression)
      statement;
```

Se `expression` è diversa da zero (TRUE) allora viene eseguito `statement`

```c
#include <stdio.h>

int x, y;

int main()
{
    /* Input the two values to be tested */

    printf("\nInput an integer value for x: ");
    scanf("%d", &x);
    printf("\nInput an integer value for y: ");
    scanf("%d", &y);

    /* Test values and print result */

    if (x == y)
        printf("x is equal to y\n");

    if (x > y)
        printf("x is greater than y\n");

    if (x < y)
        printf("x is smaller than y\n");

    return 0;
}
```



Il costrutto if –else (I)

```
if (a>b)
{
    max_ab = a;
    printf("Il massimo è a e vale %f",a);
}
else
{
    max_ab = b;
    printf("Il massimo è b e vale %f",b);
}
```

Indentazione

Il costrutto if –else (II)

```
if (a=b)
      printf("a e b sono uguali");
else
      printf("a e b sono diversi");
```


```
if (a==b)
      printf("a e b sono uguali");
else
      printf("a e b sono diversi");
```

Il costrutto if –else (III)

La parte else di un costrutto if-else è opzionale, e può quindi essere omessa. Nel caso di più costrutti if-else annidati questo genera un'ambiguità che viene risolta associando ogni else all'if **più interno** che ne è privo.

```
if (n>0)
      if(a>b)
            z=a;
      else
            z=b;
```

Il costrutto if –else (IV)

Quanto visto può talvolta essere fonte di errori:

```
if (n>0)
        if(a>b)
                z=a;
else
        printf("n non è positivo");
```

Il messaggio "n non è positivo" in questo caso viene stampato quando è verificato il primo if (n>0) e b è maggiore o uguale ad a !!!

Il costrutto if-else (V)

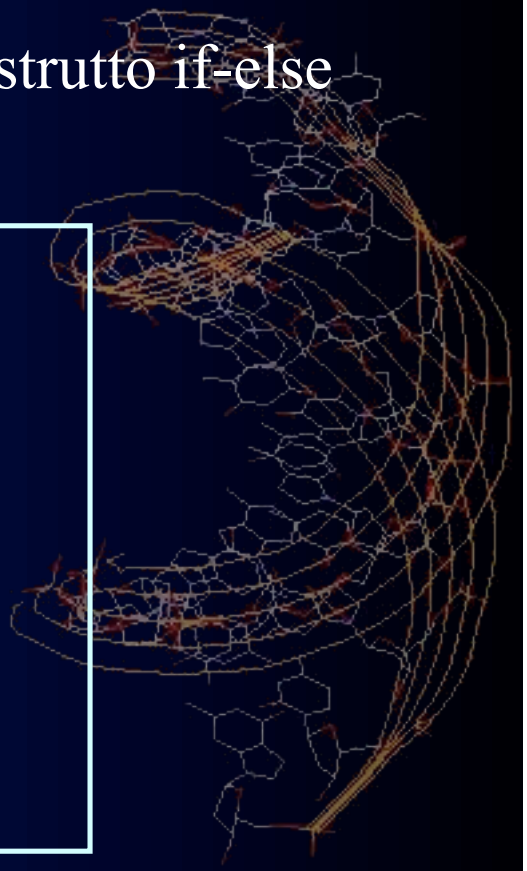
```
if (n>0)
{
        if(a>b)
                z=a;
}
else
        printf("n non è positivo");
```

Insomma USATE LE PARENTESI !!!!!

Il costrutto else if

Quando la scelta è fra più di due alternative il costrutto if-else può essere esteso utilizzando il costrutto else if :

```
if (n>0)
      printf("n è positivo");

else if (n==0)
      printf("n è nullo");
else
      printf("n è negativo");
```



Le alternative vengono esaminate in sequenza, e l'ultimo else (opzionale) corrisponde alla scelta "nessuna delle precedenti".

Sintassi

Caso 1:

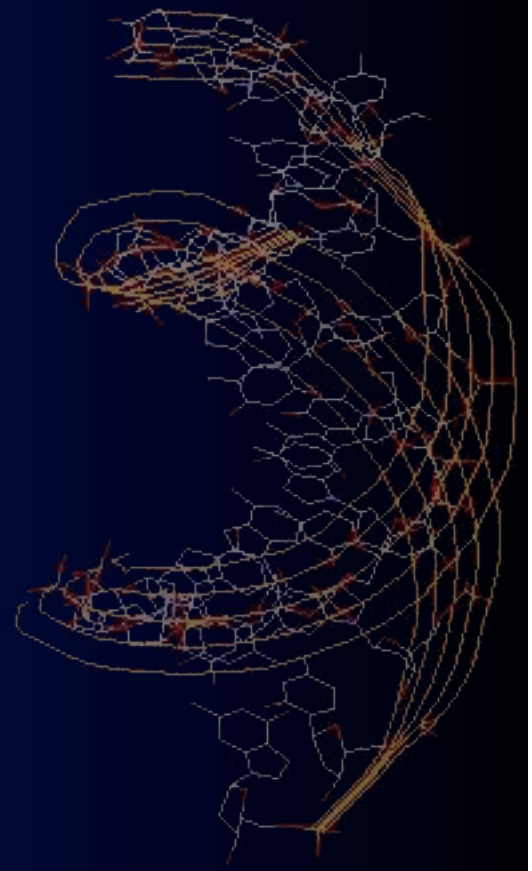
```
if( expression )
    statement1;
next_statement;
```

Caso 2:

```
if( expression )
    statement1;
else
    statement2;
next_statement;
```

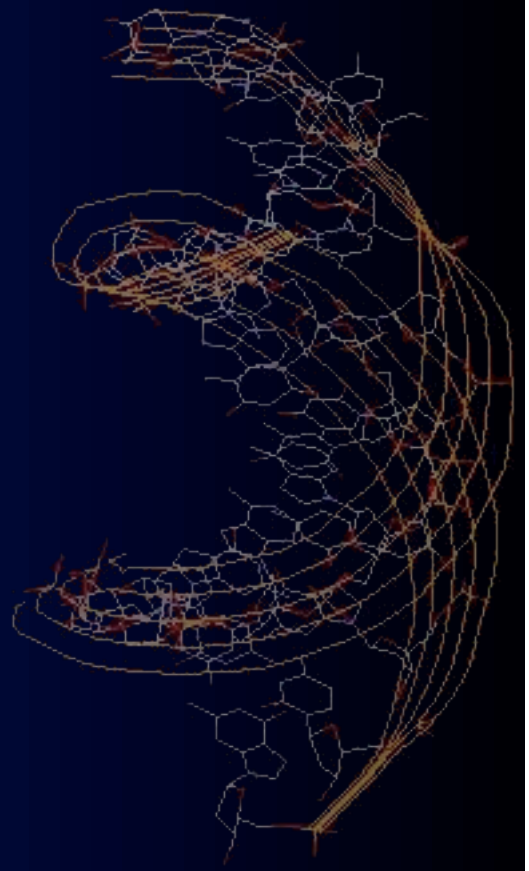
Caso 3

```
if( expression1 )
    statement1;
else if( expression2 )
    statement2;
else
    statement3;
next_statement;
```

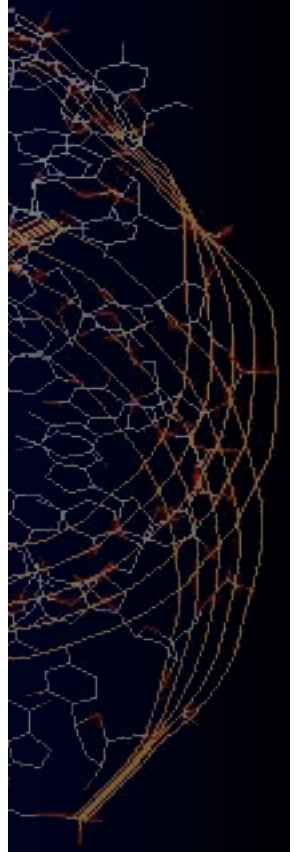


BUG BUSTER

```
/* a program with problems . . . */
#include <stdio.h>
int x= 1:
int main()
{
   if( x = 1);
      printf(" x equals 1" );
   otherwise
      printf(" x does not equal 1");
   return 0;
}
```



```c
/* Name: Find_nbr.c
* Purpose: This program picks a random number and then
* lets the user try to guess it
* Returns: Nothing
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

 #define NO 0
 #define YES 1

 void main( void )
 {
 int guess_value = -1;
 int number;
 int nbr_of_guesses;
 int done = NO;

 printf("\n\nGetting a Random number\n");

 /* use the time to seed the random number generator */
 srand( (unsigned) time( NULL ) );
 number = rand();

 nbr_of_guesses = 0;
 while ( done == NO )
 {
 printf("\nPick a number between 0 and %d> ", RAND_MAX);
 scanf( "%d", &guess_value ); /* Get a number */

 nbr_of_guesses++;
```



```
if ( number == guess_value )
{
done = YES;
}
else
if ( number < guess_value )
{
printf("\nYou guessed high!");
}
else
{
printf("\nYou guessed low!");
}
}

printf("\n\nCongratulations! You guessed right in %d Guesses!",
nbr_of_guesses);
printf("\n\nThe number was %d\n\n", number);
}
```