Informatica per sopravvivere

(ovvero: nanoLinux da principio)

Massimo Piai ⟨*pxam67 (@) virgilio·it*⟩

2006.02.19

Massimo Piai è un matematico appassionato di informatica, che ha trovato nel software libero e nella libertà delle informazioni l'unica possibilità di sviluppare tale passione. I suoi campi di interesse attuali sono la matematica e le scienze, la diffusione della Cultura Informatica, la didattica e la pedagogia.

Il presente lavoro è stato realizzato utilizzando Alml, il sistema di composizione SGML realizzato da Daniele Giacomini per la gestione dei suoi *Appunti di informatica libera*.

Indice generale

Introduzione

Questo lavoro è una raccolta di diverse piccole guide alla sopravvivenza in campo informatico. In pratica costituisce un resoconto dell'esperienza dell'autore nel campo dell'insegnamento di e con le nuove tecnologie mediante l'uso di strumenti liberi.[1]

Si tratta di un lavoro in corso d'opera, perciò è sottoposto a frequenti aggiornamenti e modifiche.

Per verificare la correttezza degli esempi[2] proposti, quando è necessario utilizzare un ambiente di lavoro specifico, l'autore del presente lavoro fa riferimento a nanoLinux,[3] un sistema GNU/Linux autoavviabile creato da Daniele Giacomini a partire dalla distribuzione GNU/Linux Debian.

Diverse parti del presente lavoro derivano più o meno esplicitamente da spunti - sia stilistici che di contenuto - ricavati dagli *Appunti di informatica libera* di Daniele Giacomini, come spiegato nelle informazioni legali all'inizio del presente lavoro. Inoltre - sebbene gran parte dei concetti qui esposti siano standard e quindi indipendenti dalla piattaforma hardware e software utilizzata - gli esempi illustrati sono stati verificati su un sistema GNU/Linux e su una piattaforma hardware i386 (Intel).

Il presente lavoro è stato realizzato utilizzando Alml,[4] il sistema di composizione SGML realizzato da Daniele Giacomini per la gestione dei suoi *Appunti di informatica libera*.

Convenzioni, note stilistiche e suggerimenti vari

Listati numerati

Questo è un lavoro di carattere tecnico, di conseguenza contiene molti listati (completi o frammenti) che si riferiscono al contenuto di file di testo, come nel caso del codice sorgente dei programmi scritti nei diversi linguaggi di programmazione.

A volte i listati sono numerati, soprattutto se molto lunghi, in modo da favorirne il commento nel testo o il riferimento da parte di altri documenti. Se si desidera utilizzare i listati, dopo averli in qualche modo estratti, risulta necessario eliminare la numerazione. Ciò è possibile, utilizzando il programma SED, con il comando[5]:

```
sed -e "s/\(^........\|^.......$\)//g" file
```

come nel seguente esempio:

```
$ cat tmp/bibbo [ Invio ]

 1 Questo è
 2 un file di testo...
 3
 4 È anche numerato.
 5 ...
 6 La riga seguente è intenzionalmente lasciata vuota.
 7
 8 ...
 9 ...
10 *CIAO A TUTTI!*
```

```
$ sed -e "s/\(^........\|^.......$\)//g" tmp/bibbo > tmp/bibbo2 [ Invio ]

$ cat tmp/bibbo2 [ Invio ]

  Questo è
  un file di testo...

  È anche numerato.
  ...
  La riga seguente è intenzionalmente lasciata vuota.

  ...
  ...
  *CIAO A TUTTI!*


$
```

Naturalmente è possibile utilizzare le funzioni di ricerca e sostituzione del proprio *editor* preferito; ad esempio con VI si può utilizzare la seguente successione di comandi:

:%s/^......//g [*Invio*]

:%s/^.//g [*Invio*]

---

[1] Tenuto conto del tristo atteggiamento comunemente incontrato in vari ambienti (purtroppo anche quello scolastico) nei confronti della libertà del software e in generale dell'informatica e delle «nuove» tecnologie, l'autore del presente lavoro ha avuto spesso la tentazione di intitolarlo polemicamente *Informatica degenerata*, in onore della cosiddetta ***Arte degenerata*** (*Entartete Kunst*). Non è escluso che ciò possa avvenire in una futura edizione!

[2] A volte gli esempi sono parte integrante del testo principale, altre volte invece lo interrompono (ossia costituiscono unità abbastanza indipendenti dal testo principale, tant'è vero che presentano una didascalia con numerazione); in quest'ultimo caso, per indicare la ripresa del testo principale, si utilizzerà come segnale il carattere ►.

[3] **nanoLinux** GNU GPL; i singoli applicativi sono sottoposti eventualmente alle loro condizioni specifiche

[4] **Alml** GNU GPL

[5] I caratteri '**.**' denotano quanto precede, in ciascuna riga da trattare, il testo effettivo; ciò può essere costituito da un numero «quasi» fisso di caratteri reali; negli esempi mostrati si tratta di 8 oppure 7 caratteri, in altri casi tali quantità possono essere diverse, e gli esempi devono essere conseguentemente adattati.

Parte i

Python per sopravvivere

In questa parte verranno introdotti i principali elementi della sintassi e dell'idioma del linguaggio Python,[1] a partire da semplici esempi di programmazione. Il principale vantaggio di Python per un uso didattico è che la sintassi è molto semplice, tanto da far sembrare i programmi scritti in Python quasi come se fossero stati scritti in uno pseudolinguaggio.

Ma Python non è un linguaggio «giocattolo»: si tratta infatti di un ambiente per lo sviluppo rapido di prototipi e anche di applicazioni complesse, è dotato di un'ampia gamma di moduli per scopi specializzati (anche sviluppati da terze parti), è stato adattato da più soggetti ed a diverse piattaforme, è stato utilizzato per lo sviluppo di alcune «killer application», ad esempio Google. Si tratta inoltre di un linguaggio orientato agli oggetti fin dalla progettazione, il che può sembrare vantaggioso a qualcuno.

Come prerequisito per la lettura è necessario essere stati introdotti ai rudimenti della programmazione strutturata, mediante linguaggi specifici oppure pseudocodifica.

---

[1] **Python** software con licenza GPL-compatibile

Primo esempio di programma Python

Per cominciare a cogliere un'idea delle caratteristiche[1] di Python, si consideri il problema della somma di due numeri positivi espressa attraverso il concetto dell'incremento unitario: ***n***+***m*** equivale a incrementare ***m***, di un'unità, per ***n*** volte, oppure incrementare ***n*** per ***m*** volte. L'algoritmo risolutivo è banale, ma utile per apprendere il funzionamento dei cicli; il listato 1.1 presenta la soluzione tramite pseudocodifica.

Listato 1.1. Somma ciclica (pseudocodifica).

```
SOMMA (X, Y)

    LOCAL Z INTEGER
    LOCAL I INTEGER

    Z := X
    FOR I := 1; I <= Y; I++
        Z++
    END FOR

    RETURN Z

END SOMMA
```

Tenendo presente che in Python l'incremento unitario della variabile '***variabile***'; si esprime con l'idioma

```
variabile += 1
```

il listato 1.2 traduce l'algoritmo in un programma Python completo e commentato.

Listato 1.2. Primo esempio in Python.

```
#!/usr/bin/python
##
## somma.py <x> <y>
## Somma esclusivamente valori positivi.
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# somma(<x>, <y>)
#
def somma(x, y):
  z = x
  for i in range(y):
    z += 1
  return z
#
# Inizio del programma.
#
```

```
x = int(sys.argv[1])
y = int(sys.argv[2])
z = somma(x, y)
print x, "+", y, "=", z
```

Si noti in particolare la compattezza del programma Python: solo dieci righe «utili», contro le sedici dell'equivalente programma in Perl (listato 1.3) e addirittura ventidue dell'equivalente programma in C (listato 1.4).

Listato 1.3. Somma ciclica (Perl).

```
#!/usr/bin/perl
##
## somma.pl <x> <y>
## Somma esclusivamente valori positivi.
##
#
# &somma (<x>, <y>)
#
sub somma
{
    local ($x) = $_[0];
    local ($y) = $_[1];
    #
    local ($z) = $x;
    local ($i);
    #
    for ($i = 1; $i <= $y; $i++)
      {
        $z++;
      }
    #
    return $z;
}
#
# Inizio del programma.
#
$x = $ARGV[0];
$y = $ARGV[1];
#
$z = &somma ($x, $y);
#
print "$x + $y = $z\n";
#
```

Listato 1.4. Somma ciclica (C).

```
/* ================================================================= */
/* somma <x> <y>                                                     */
/* Somma esclusivamente valori positivi.                             */
/* ================================================================= */

#include <stdio.h>

/* ================================================================= */
/* somma (<x>, <y>)                                                  */
/* ----------------------------------------------------------------- */
int somma (int x, int y)
{
    int z = x;
    int i;

    for (i = 1; i <= y; i++)
      {
        z++;
      };

    return z;
}

/* ================================================================= */
/* Inizio del programma.                                             */
/* ----------------------------------------------------------------- */
int main (int argc, char *argv[])
{
    int x;
    int y;
    int z;

    /* Converte le stringhe ottenute dalla riga di comando in
       numeri interi e li assegna alle variabili x e y.           */
    sscanf (argv[1], "%d", &x);
    sscanf (argv[2], "%d", &y);

    z = somma (x, y);

    printf ("%d + %d = %d\n", x, y, z);

    return 0;
}
```


[1] Salvo diverso avviso, nel seguito si farà riferimento alla versione 2.3 del linguaggio.

Capitolo 2

Studio dettagliato del precedente esempio

Si procede ora ad analizzare il testo del semplice programma Python visto in precedenza (listato 1.2).

La riga

```
#!/usr/bin/python
...
```

come di consueto indica che quanto segue deve essere considerata una successione di comandi da «dare in pasto» all'interprete indicato, in questo caso '**`python`**' ossia l'eseguibile corrispondente a Python.

Le righe successive che inizino con il carattere cancelletto, '**`#`**', vanno considerati dei commenti ad uso del lettore e vengono ignorate dall'interprete.

La riga

```
...
import sys
...
```

serve a importare nel programma quanto contenuto (definizioni di funzioni, variabili, eccetera) nel ***modulo*** '**`sys`**', il quale è parte della libreria standard di Python.

Per accedere alle componenti del modulo si ricorre alla ***notazione puntata***, come si vedrà nel seguito.

La riga

```
...
def somma(x, y):
...
```

costituisce l'intestazione di una funzione, mediante la quale si definisce la funzione stessa, in questo caso '**`somma`**'.

Fra parentesi vanno indicati i parametri formali della funzione; l'intestazione termina obbligatoriamente con i due punti ('**`:`**').

Il blocco

```
...
  z = x
  for i in range(y):
    z += 1
  return z
...
```

costituisce il corpo della funzione '**somma**'.

Come si può notare l'appartenenza di un blocco al corpo di una funzione è stabilita dall'allineamento delle righe successive: la prima riga del corpo deve rientrare a destra di almeno uno spazio rispetto all'intestazione, e le righe del blocco devono essere allineate con la prima; il corpo della funzione termina con la prima riga che sia allineata con l'intestazione oppure rientri a sinistra rispetto a questa. Questo sistema di rientri costituisce la convenzione mediante cui in Python si raggruppano le righe in blocchi e si indica il controllo di un istruzione su una riga o blocco; in generale lo schema è:

```
istruzione_che_controlla:
    istruzione_controllata
    istruzione_controllata
    istruzione_controllata
    ...

istruzione_non_controllata
```

Si noti in particolare la presenza del carattere '**:**' nella riga di controllo.

La riga

```
...
  z = x
...
```

rappresenta un'istruzione di assegnamento alla variabile locale '**z**'.

La riga

```
...
  for i in range(y):
...
```

è l'istruzione di controllo di un ciclo enumerativo.

Si tratta di un tipico idioma Python: il costrutto

```
for variabile in lista:
  ...
```

indica che durante l'esecuzione del ciclo la variabile assume in successione i valori di una ***lista***, quindi il ciclo ha termine; l'idioma

```
range(valore)
```

restituisce una lista di valori da 0 a ***valore***−1.

La riga

```
...
    z += 1
...
```

è l'idioma Python che descrive l'incremento unitario di una variabile.

La riga

```
...
  return z
...
```

permette alla funzione '**somma**' di restituire il valore della variabile locale '**z**' al chiamante.

Dalla riga

```
...
x = int(sys.argv[1])
...
```

inizia il corpo principale del programma; mediante la notazione '**sys.argv**' si accede alla variabile '**argv**' definita nel modulo '**sys**'; la variabile '**argv**' contiene una lista di valori corrispondenti rispettivamente al nome del programma eseguito ('**sys.argv[0]**') e agli argomenti forniti allo stesso sulla linea di comando ('**sys.argv[1]**', '**sys.argv[2]**', eccetera). Poiché gli argomenti vengono passati al programma sottoforma di stringhe, è necessario convertirli nel caso in cui debbano essere intesi diversamente, per esempio come valori numerici interi; a tal fine si utilizza proprio la notazione

```
int(valore_non_intero)
```

la quale restituisce per l'appunto il valore intero corrispondente all'argomento fornito (sempre che ciò sia possibile); il valore ottenuto viene poi assegnato alla variabile globale '**x**'. A questo punto il significato della riga successiva dovrebbe risultare evidente.

La riga

```
...
z = somma(x, y)
...
```

rappresenta la chiamata della funzione '**somma**' precedentemente definita, con parametri '**x**' e '**y**', e successivamente l'assegnamento del valore restituito alla funzione alla variabile globale '**z**'.

Infine, la riga

```
...
print x, "+", y, "=", z
...
```

serve a inviare allo standard output (ossia generalmente lo schermo del terminale) i risultati

dell'elaborazione; come si vede, la parola chiave '**`print`**' può essere seguita da più argomenti, variabili o costanti di tipo diverso, separati dal carattere virgola ('**`,`**').

Un modo alternativo per generare lo stesso output è quello di utilizzare l'***operatore di formato*** ('**`%`**'), la cui sintassi rivela un evidente eredità dal linguaggio C:

```
print "%d + %d = %d" % (x, y, z)
```

Per concludere, ecco un esempio di esecuzione del programma Python da riga di comando:

```
$ ls -l somma.py [ Invio ]

  -rw-r--r--    1 max2     max2          349 2004-09-19 22:06 somma.py


$ chmod +x somma.py [ Invio ]

$ ls -l somma.py [ Invio ]

  -rwxr-xr-x    1 max2     max2          349 2004-09-19 22:06 somma.py


$ ./somma.py 23 34 [ Invio ]

  23 + 34 = 57


$
```

Capitolo 3

Ulteriori esempi di programmi Python

Nelle sezioni seguenti sono descritti alcuni problemi elementari attraverso cui si insegnano le tecniche di programmazione ai principianti. Assieme ai problemi vengono proposte le soluzioni in forma di programma Python. Per le soluzioni in forma di pseudocodifica si rimanda agli *Appunti di informatica libera* di Daniele Giacomini.

3.1 Problemi elementari di programmazione

3.1.1 Somma attraverso incremento unitario: versione con ciclo iterativo

```
...
def somma(x, y):
  z = x
  i = 1
  while i <= y:
    z += 1
    i += 1
  return z
...
```

3.1.2 Moltiplicazione di due numeri positivi attraverso la somma

La moltiplicazione di due numeri positivi, può essere espressa attraverso il concetto della somma: ***n**m*** equivale a sommare ***m*** volte ***n***, oppure ***n*** volte ***m***. L'algoritmo risolutivo è banale, ma utile per apprendere il funzionamento dei cicli.

Listato 3.2. Moltiplicazione ciclica.

```
#!/usr/bin/python
##
## moltiplica.py <x> <y>
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# moltiplica(<x>, <y>)
#
def moltiplica(x, y):
  z = 0
  for i in range(y):
    z = z+x
  return z
#
# Inizio del programma.
#
x = int(sys.argv[1])
y = int(sys.argv[2])
z = moltiplica(x, y)
print x, "*", y, "=", z
```

Nel listato 3.2 viene mostrata una soluzione per mezzo di un ciclo enumerativo. Il ciclo viene ripetuto '**y**' volte, incrementando la variabile '**z**' del valore di '**x**'. Alla fine, '**z**' contiene il

risultato del prodotto di '**x**' per '**y**'. Il frammento seguente mostra invece la traduzione del ciclo enumerativo in un ciclo iterativo:

```
...
def moltiplica(x, y):
  z = 0
  i = 1
  while i <= y:
    z = z+x
    i += 1
  return z
...
```

3.1.3 Divisione intera tra due numeri positivi

La divisione di due numeri positivi, può essere espressa attraverso la sottrazione: ***n***/***m*** equivale a sottrarre ***m*** da ***n*** fino a quando ***n*** diventa inferiore di ***m***. Il numero di volte in cui tale sottrazione ha luogo, è il risultato della divisione.

Listato 3.4. Divisione ciclica.

```
#!/usr/bin/python
##
## dividi.py <x> <y>
## Divide esclusivamente valori positivi.
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# dividi(<x>, <y>)
#
def dividi(x, y):
  z = 0
  i = x
  while i >= y:
    i = i-y
    z += 1
  return z
#
# Inizio del programma.
#
x = int(sys.argv[1])
y = int(sys.argv[2])
z = dividi(x, y)
print "Divisione intera -> %d/%d = %d" % (x, y, z)
```

Il listato 3.4 realizza l'algoritmo; si noti anche l'uso dell'operatore di formato.

3.1.4 Elevamento a potenza

L'elevamento a potenza, utilizzando numeri positivi, può essere espresso attraverso il concetto della moltiplicazione: ***n*****m*** equivale a moltiplicare ***m*** volte ***n*** per se stesso.

Listato 3.5. Potenza ciclica.

```
#!/usr/bin/python
##
## exp.py <x> <y>
## Eleva a potenza.
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# exp(<x>, <y>)
#
def exp(x, y):
  z = 1
  for i in range(y):
    z = z*x
  return z
#
# Inizio del programma.
#
x = int(sys.argv[1])
y = int(sys.argv[2])
z = exp(x, y)
print "%d ** %d = %d" % (x, y, z)
```

Nel listato 3.5 viene mostrata una soluzione per mezzo di un ciclo enumerativo. Il ciclo viene ripetuto '**y**' volte; ogni volta la variabile '**z**' viene moltiplicata per il valore di '**x**', a partire da '**1**'. Alla fine, '**z**' contiene il risultato dell'elevamento di '**x**' a '**y**'. Il frammento seguente mostra invece la traduzione del ciclo enumerativo in un ciclo iterativo:

```
...
def exp(x, y):
  z = 1
  i = 1
  while i <= y:
    z = z*x
    i += 1
  return z
...
```

Il frammento seguente mostra una soluzione ricorsiva:

```
...
def exp(x, y):
  if x == 0:
    return 0
```

```
  elif y == 0:
    return 1
  else:
    return x*exp(x, y-1)
...
```

3.1.5 Radice quadrata

Il calcolo della parte intera della radice quadrata di un numero si può fare per tentativi, partendo da 1, eseguendo il quadrato fino a quando il risultato è minore o uguale al valore di partenza di cui si calcola la radice.

Il listato 3.8 realizza l'algoritmo.

Listato 3.8. Radice quadrata ciclica.

```
#!/usr/bin/python
##
## radice.py <x>
## Radice quadrata.
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# radice(<x>)
#
def radice(x):
  z = 0
  t = 0
  while True:
    t = z*z
    if t > x:
      #
      # E' stato superato il valore massimo.
      #
      z -= 1
      return z
    z += 1
  #
  # Teoricamente, non dovrebbe mai arrivare qui.
  #
#
# Inizio del programma.
#
x = int(sys.argv[1])
z = radice(x)
print "radq(%d) = %d" % (x, z)
```

3.1.6 Fattoriale

Il fattoriale è un valore che si calcola a partire da un numero positivo. Può essere espresso come il prodotto di ***n*** per il fattoriale di ***n***−1, quando ***n*** è maggiore di 1, mentre equivale a 1 quando ***n*** è uguale a 1. In pratica, ***n***! = ***n*** * (***n***−1) * (***n***−2)... *1.

Listato 3.9. Fattoriale.

```
#!/usr/bin/python
##
## fatt.py <x>
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# fatt(<x>)
#
def fatt(x):
  i = x-1
  while i > 0:
    x = x*i
    i -= 1
  return x
#
# Inizio del programma.
#
x = int(sys.argv[1])
fatt = fatt(x)
print "%d! = %d" % (x, fatt)
```

La soluzione mostrata nel listato 3.9 fa uso di un ciclo iterativo in cui l'indice '**i**', che inizialmente contiene il valore di '**x-1**', viene usato per essere moltiplicato al valore di '**x**', riducendolo ogni volta di un'unità. Quando '**i**' raggiunge lo '**0**', il ciclo termina e '**x**' contiene il valore del fattoriale. L'esempio seguente mostra invece una soluzione ricorsiva che dovrebbe risultare più intuitiva:

```
...
def fatt(x):
  if x == 1:
    return 1
  else:
    return x*fatt(x-1)
...
```

3.1.7 Massimo comune divisore

Il massimo comune divisore tra due numeri può essere ottenuto sottraendo a quello maggiore il valore di quello minore, fino a quando i due valori sono uguali. Quel valore è il massimo comune divisore.

Il listato 3.11 realizza l'algoritmo.

Listato 3.11. Massimo comune divisore.

```
#!/usr/bin/python
##
## mcd.py <x> <y>
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# mcd(<x>, <y>)
#
def mcd(x, y):
  while x != y:
    if x > y:
      x = x-y
    else:
      y = y-x
  return x
#
# Inizio del programma.
#
x = int(sys.argv[1])
y = int(sys.argv[2])
z = mcd(x, y)
print "Il massimo comune divisore di %d e %d e' %d" %  (x, y, z)
```

3.1.8 Numero primo

Un numero intero è numero primo quando non può essere diviso per un altro intero diverso dal numero stesso e da 1, generando un risultato intero.

Il listato 3.12 realizza l'algoritmo.

Listato 3.12. Numero primo.

```
#!/usr/bin/python
##
## primo.py <x>
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
```

```
# primo(<x>)
#
def primo(x):
  primo = True
  i = 2
  while i < x and primo:
    j = x/i
    j = x-(j*i)
    if j == 0:
      primo = False
    else:
      i += 1
  return primo
#
# Inizio del programma.
#
x = int(sys.argv[1])
if primo(x):
  print x, "e‘ un numero primo"
else:
  print x, "non e‘ un numero primo"
```

3.2 Scansione di array

Nelle sezioni seguenti sono descritti alcuni problemi legati alla scansione di array. Assieme ai problemi vengono proposte le soluzioni in forma di programmi Python.

3.2.1 Ricerca sequenziale

La ricerca di un elemento all'interno di un array disordinato può avvenire solo in modo sequenziale, cioè controllando uno per uno tutti gli elementi, fino a quando si trova la corrispondenza cercata. La tabella 3.13 presenta la descrizione delle variabili più importanti che appaiono nei programmi Python successivi.

Tabella 3.13. Ricerca sequenziale: variabili utilizzate.

Variabile	Descrizione
`lista`	È l'array su cui effettuare la ricerca.
`x`	È il valore cercato all'interno dell'array.
`a`	È l'indice inferiore dell'intervallo di array su cui si vuole effettuare la ricerca.
`z`	È l'indice superiore dell'intervallo di array su cui si vuole effettuare la ricerca.

Il listato 3.14 presenta un esempio di programma Python che risolve il problema in modo iterativo.

Listato 3.14. Ricerca sequenziale.

```
#!/usr/bin/python
##
## ricercaseq.py <elemento-cercato> <valore>...
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# ricercaseq(<lista>, <elemento>, <inizio>, <fine>)
#
def ricercaseq(lista, x, a, z):
  for i in range(a, z+1):
    if x == lista[i]:
      return i
  return -1
#
# Inizio del programma.
#
x = sys.argv[1]
lista = sys.argv[2:]
i = ricercaseq(lista, x, 0, len(lista)-1)
if i != -1:
  print "L'elemento", x, "si trova nella posizione", i
else:
  print "L'elemento", x, "non e' stato trovato"
```

L'algoritmo usato dovrebbe risultare abbastanza chiaro.

Per quanto concerne le caratteristiche del linguaggio usate, si noti che Python offre nativamente supporto per le liste, le quali sono strutture dati più complesse degli array, ma che ovviamente possono essere utilizzate per simularli, proprio come nell'esempio; le liste hanno gli indici che vanno da '**0**' a '**len(*lista*)-1**', ove la funzione '**len**' restituisce il numero di elementi della lista '***lista***'; per accedere ai singoli elementi di una lista si usa la notazione

***lista*[*indice*]**

inoltre è possibile estrarre una sottolista mediante la notazione

***lista*[*indice_iniziale*:*indice_finale*]**

eventualmente tralasciando uno dei due indici (o anche entrambi), che quindi assumono valori predefiniti (rispettivamente '**0**' e '**len(*lista*)**'; si tenga presente che in tale notazione l'elemento corrispondente all'indice finale si intende da escludersi.

Esiste anche una soluzione ricorsiva che viene mostrata nel frammento seguente:

```
...
def ricercaseq(lista, x, a, z):
  if a > z:
```

```
    return -1
  elif x == lista[a]:
    return a
  else:
    return ricercaseq(lista, x, a+1, z)
...
```

3.2.2 Ricerca binaria

La ricerca di un elemento all'interno di un array ordinato può avvenire individuando un elemento centrale: se questo corrisponde all'elemento cercato, la ricerca è terminata, altrimenti si ripete nella parte di array precedente o successiva all'elemento, a seconda del suo valore e del tipo di ordinamento esistente.

Il problema posto in questi termini è ricorsivo. Il programma mostrato nel listato 3.16 utilizza le stesse variabili già descritte per la ricerca sequenziale.

Listato 3.16. Ricerca binaria.

```
#!/usr/bin/python
##
## ricercabin.py <elemento-cercato> <valore>...
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# ricercabin(<lista>, <elemento>, <inizio>, <fine>)
#
def ricercabin(lista, x, a, z):
  #
  # Determina l'elemento centrale.
  #
  m = (a+z)/2
  if m < a:
    #
    # Non restano elementi da controllare: l'elemento cercato
    # non c'e'.
    #
    return -1
  elif x < lista[m]:
    #
    # Si ripete la ricerca nella parte inferiore.
    #
    return ricercabin(lista, x, a, m-1)
  elif x > lista[m]:
    #
    # Si ripete la ricerca nella parte superiore.
    #
    return ricercabin(lista, x, m+1, z)
  else:
    #
```

```
    # m rappresenta l'indice dell'elemento cercato.
    #
    return m
#
# Inizio del programma.
#
x = sys.argv[1]
lista = sys.argv[2:]
i = ricercabin(lista, x, 0, len(lista)-1)
if i != -1:
  print "L'elemento", x, "si trova nella posizione", i
else:
  print "L'elemento", x, "non e' stato trovato"
```

3.3 Problemi classici di programmazione

Nelle sezioni seguenti sono descritti alcuni problemi classici attraverso cui si insegnano le tecniche di programmazione. Assieme ai problemi vengono proposte le soluzioni in forma di programma Python.

3.3.1 Bubblesort

Il Bubblesort è un algoritmo relativamente semplice per l'ordinamento di un array, in cui ogni scansione trova il valore giusto per l'elemento iniziale dell'array stesso. Una volta trovata la collocazione di un elemento, si ripete la scansione per il segmento rimanente di array, in modo da collocare un altro valore. Il testo del programma dovrebbe chiarire il meccanismo. La tabella 3.17 presenta la descrizione delle variabili più importanti utilizzate dal programma.

Tabella 3.17. Bubblesort: variabili utilizzate.

Variabile	Descrizione
`lista`	È l'array da ordinare.
`a`	È l'indice inferiore del segmento di array da ordinare.
`z`	È l'indice superiore del segmento di array da ordinare.

Nel listato 3.18 viene mostrata una soluzione iterativa.

Listato 3.18. Bubblesort.

```
#!/usr/bin/python
##
## bsort.py <valore>...
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# bsort(<lista>, <inizio>, <fine>)
```

```
#
def bsort(lista, a, z):
  #
  # Inizia il ciclo di scansione dell'array.
  #
  for j in range(a, z):
    #
    # Scansione interna dell'array per collocare nella posizione
    # j l'elemento giusto.
    #
    for k in range(j+1, z+1):
      if lista[k] < lista[j]:
        #
        # Scambia i valori
        #
        scambio = lista[k]
        lista[k] = lista[j]
        lista[j] = scambio
#
# Inizio del programma.
#
lista = sys.argv[1:]
bsort(lista, 0, len(lista)-1)
for elemento in lista:
  print elemento,
```

Vale la pena di osservare nelle ultime due righe alcuni aspetti idiomatici di Python: avendo a disposizione una lista è possibile utilizzare i suoi elementi come indici di un ciclo enumerativo utilizzando la consueta parola chiave '**`in`**'; inoltre, se si intende che l'output non vada a capo ma prosegua sulla medesima riga, si può usare il carattere vigola ('**`,`**') in coda all'istruzione '**`print`**'.

Segue la funzione '**`bsort`**' in versione ricorsiva:

```
...
def bsort(lista, a, z):
  if a < z:
    #
    # Scansione interna dell'array per collocare nella posizione
    # a l'elemento giusto.
    #
    for k in range(a+1, z+1):
      if lista[k] < lista[a]:
        #
        # Scambia i valori
        #
        scambio = lista[k]
        lista[k] = lista[a]
        lista[a] = scambio
    bsort(lista, a+1, z)
...
```

3.3.1.1 Alcune osservazioni aggiuntive

Si tenga presente che in questa sezione, come in quelle sugli algoritmi di ricerca, l'istruzione di lettura degli argomenti della linea di comando non prevede la trasformazione da stringa a intero; questo perché, in generale, ci si può aspettare che una lista sia costituita da elementi non numerici.

Tuttavia, si consideri la seguente situazione:

```
$ ./bsort.py 3 5 8 2 9 4512 7 67431 3 6 3 [ Invio ]

  2 3 3 3 4512 5 6 67431 7 8 9
```

Ovviamente non è quello che ci si aspetta; in realtà, l'output è comprensibile se si tiene presente che gli argomenti vengono trattati come stringhe, perciò la lista viene ordinata secondo l'ordine lessicografico basato sul codice ASCII (in pratica l'ordine alfabetico esteso).

Se si vuole correggere il comportamanto del programma, è possibile sostituire la riga

```
...
lista = sys.argv[1:]
...
```

con la riga

```
...
lista = map(int, sys.argv[1:])
...
```

Con l'occasione, si noti un'ulteriore interessante aspetto idiomatico di Python: è possibile applicare una funzione per trasformare tutti i membri di una lista utilizzando il costrutto:

```
map(funzione, lista)
```

Si invita il lettore interessato a consultare la documentazione di Python per ulteriori aspetti riguardanti la gestione delle liste.

Ecco il comportamento del programma modificato:

```
$ ./bsort.py 3 5 8 2 9 4512 7 67431 3 6 3 [ Invio ]

  2 3 3 3 5 6 7 8 9 4512 67431
```

Vale la pena notare che, così modificato, il programma non funziona se gli argomenti passatigli non possono essere interpretati come numeri interi.

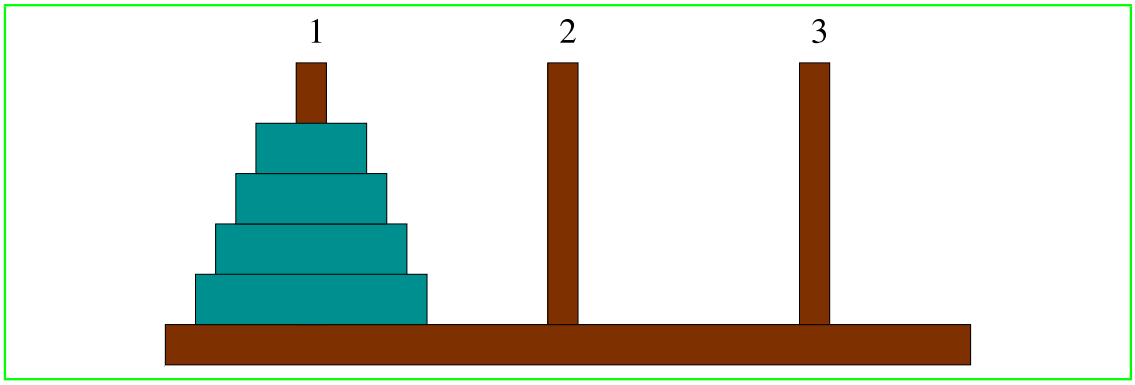
3.3.2 Torre di Hanoi

La torre di Hanoi è un gioco antico: si compone di tre pioli identici conficcati verticalmente su una tavola e di una serie di anelli di larghezze differenti. Gli anelli sono più precisamente dei dischi con un foro centrale che permette loro di essere infilati nei pioli.

Il gioco inizia con tutti gli anelli collocati in un solo piolo, in ordine, in modo che in basso ci sia l'anello più largo e in alto quello più stretto. Si deve riuscire a spostare tutta la pila di anelli in un dato piolo muovendo un anello alla volta e senza mai collocare un anello più grande sopra uno più piccolo.

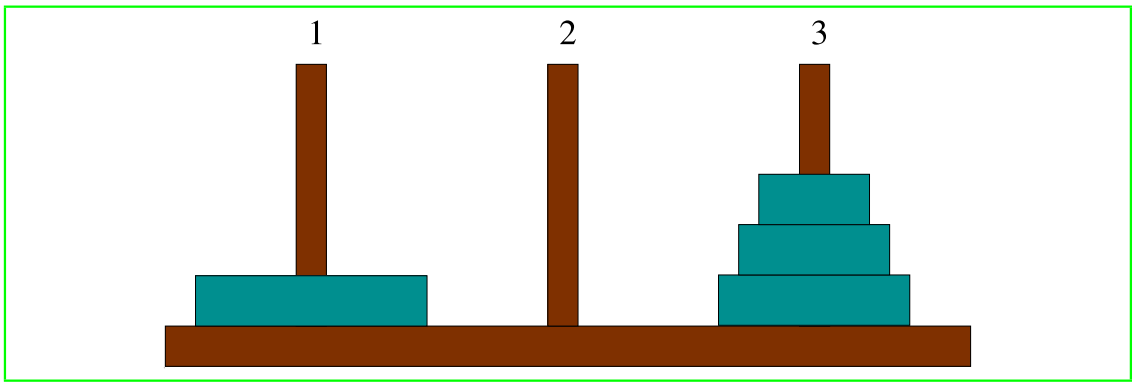
La figura 3.24 illustra la situazione iniziale della torre di Hanoi all'inizio del gioco.

Figura 3.24. Situazione iniziale della torre di Hanoi all'inizio del gioco.



Nella figura 3.24 gli anelli appaiono inseriti sul piolo 1; si supponga che questi debbano essere spostati sul piolo 2. Si può immaginare che tutti gli anelli, meno l'ultimo, possano essere spostati in qualche modo corretto, dal piolo 1 al piolo 3, come nella situazione della figura 3.25.

Figura 3.25. Situazione dopo avere spostato $n-1$ anelli.



A questo punto si può spostare l'ultimo anello rimasto (l'n-esimo), dal piolo 1 al piolo 2; quindi, come prima, si può spostare in qualche modo il gruppo di anelli posizionati attualmente nel piolo 3, in modo che finiscano nel piolo 2 sopra l'anello più grande.

Pensando in questo modo, l'algoritmo risolutivo del problema deve essere ricorsivo e potrebbe essere gestito da un'unica funzione che può essere chiamata opportunamente '**`hanoi`**', i cui parametri sono presentati nella tabella 3.26.

Tabella 3.26. Parametri sella funzione '**hanoi**'.

Parametro	Descrizione
`n`	È la dimensione della torre espressa in numero di anelli: gli anelli sono numerati da 1 a n.
`p1`	È il numero del piolo su cui si trova inizialmente la pila di n anelli.
`p2`	È il numero del piolo su cui deve essere spostata la pila di anelli.
`6-p1-p2`	È il numero dell'altro piolo. Funziona così se i pioli sono numerati da 1 a 3.

Il listato 3.27 presenta il programma Python con funzione ricorsiva per la soluzione del problema.

Listato 3.27. Torre di Hanoi.

```
#!/usr/bin/python
##
## hanoi.py <n-anelli> <piolo-iniziale> <piolo-finale>
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# hanoi(<n-anelli>, <piolo-iniziale>, <piolo-finale>)
#
def hanoi(n, p1, p2):
  if n > 0:
    hanoi(n-1, p1, 6-p1-p2)
    print "Muovi l'anello %d dal piolo %d al piolo %d" % (n, p1, p2)
    hanoi(n-1, 6-p1-p2, p2)
##
## Inizio del programma.
##
(n, p1, p2) = map(int, sys.argv[1:4])
hanoi(n, p1, p2)
```

Si colga l'occasione per osservare un'ulteriore aspetto idiomatico di Python, ossia la possibilità di assegnare «in parallelo» gli elementi di una lista a più variabili (o, come si dice in gergo Python, una ***tupla***) con una singola istruzione di assegnamento.

Tornando al problema, ecco l'analisi dell'algoritmo risolutivo: se '**n**', il numero degli anelli da spostare, è minore di '**1**', non si deve compiere alcuna azione. Se '**n**' è uguale a '**1**', le istruzioni controllate dalla struttura condizionale vengono eseguite, ma nessuna delle chiamate ricorsive fa alcunché, dato che '**n-1**' è pari a '**0**'. In questo caso, supponendo che '**n**' sia uguale a '**1**', che '**p1**' sia pari a '**1**' e '**p2**' pari a '**2**', il risultato è semplicemente:

```
Muovi l'anello 1 dal piolo 1 al piolo 2
```

Il risultato è quindi corretto per una pila iniziale consistente di un solo anello.

Se '**n**' è uguale a '**2**', la prima chiamata ricorsiva sposta un anello ('**n-1**' = '**1**') dal piolo 1 al piolo 3 (ancora assumendo che i due anelli debbano essere spostati dal primo al terzo piolo) e si sa che questa è la mossa corretta. Quindi viene stampato il messaggio che dichiara lo spostamento del secondo piolo (l'***n***-esimo) dalla posizione 1 alla posizione 2. Infine, la seconda chiamata ricorsiva si occupa di spostare l'anello collocato precedentemente nel terzo piolo, nel secondo, sopra a quello che si trova già nella posizione finale corretta.

In pratica, nel caso di due anelli che devono essere spostati dal primo al secondo piolo, appaiono i tre messaggi seguenti:

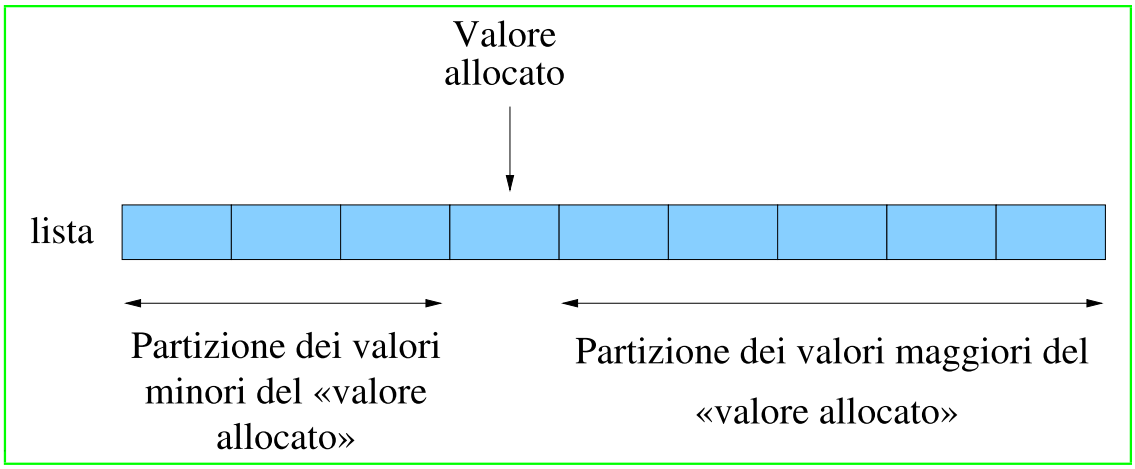
```
Muovi l'anello 1 dal piolo 1 al piolo 3
Muovi l'anello 2 dal piolo 1 al piolo 2
Muovi l'anello 1 dal piolo 3 al piolo 2
```

Nello stesso modo si potrebbe dimostrare il funzionamento per un numero maggiore di anelli.

3.3.3 Quicksort (ordinamento non decrescente)

L'ordinamento degli elementi di un array è un problema tipico che si può risolvere in tanti modi. Il Quicksort è un algoritmo sofisticato, ottimo per lo studio della gestione degli array, oltre che per quello della ricorsione. Il concetto fondamentale di questo tipo di algoritmo è rappresentato dalla figura 3.30.

Figura 3.30. Il concetto base dell'algoritmo del Quicksort: suddivisione dell'array in due gruppi disordinati, separati da un valore piazzato correttamente nel suo posto rispetto all'ordinamento.



Una sola scansione dell'array è sufficiente per collocare definitivamente un elemento (per esempio il primo) nella sua destinazione finale e allo stesso tempo per lasciare tutti gli elementi con un valore inferiore a quello da una parte, anche se disordinati, e tutti quelli con un valore maggiore, dall'altra.

In questo modo, attraverso delle chiamate ricorsive, è possibile elaborare i due segmenti dell'array rimasti da riordinare.

L'algoritmo può essere descritto grossolanamente come:

1. localizzazione della collocazione finale del primo valore, separando in questo modo i valori;
2. ordinamento del segmento precedente all'elemento collocato definitivamente;

3. ordinamento del segmento successivo all'elemento collocato definitivamente.

Viene qui indicata con '**part**' la funzione che esegue la scansione dell'array, o di un suo segmento, per determinare la collocazione finale (indice '**cf**') del primo elemento (dell'array o del segmento in questione).

Sia '**lista**' l'array da ordinare. Il primo elemento da collocare corrisponde inizialmente a '**lista[a]**' e il segmento di array su cui intervenire corrisponde a '**lista[a:z+1]**' (cioè a tutti gli elementi che vanno dall'indice '**a**' all'indice '**z**').

Alla fine della prima scansione, l'indice '**cf**' rappresenta la posizione in cui occorre spostare il primo elemento, cioè '**lista[a]**'. In pratica, '**lista[a]**' e '**lista[cf]**' vengono scambiati.

Durante la scansione che serve a determinare la collocazione finale del primo elemento, '**part**' deve occuparsi di spostare gli elementi prima o dopo quella posizione, in funzione del loro valore, in modo che alla fine quelli inferiori o uguali a quello dell'elemento da collocare si trovino nella parte inferiore e gli altri dall'altra. In pratica, alla fine della prima scansione, gli elementi contenuti in '**lista[a:cf]**' devono contenere valori inferiori o uguali a '**lista[cf]**', mentre quelli contenuti in '**lista[cf+1:z+1]**' devono contenere valori superiori.

Indichiamo con '**qsort**' la funzione che esegue il compito complessivo di ordinare l'array. Il suo lavoro consisterebbe nel chiamare '**part**' per collocare il primo elemento, continuando poi con la chiamata ricorsiva di se stessa per la parte di array precedente all'elemento collocato e infine alla chiamata ricorsiva per la parte restante di array.

Assumendo che '**part**' e le chiamate ricorsive di '**qsort**' svolgano il loro compito correttamente, si potrebbe fare un'analisi informale dicendo che se l'indice '**z**' non è maggiore di '**a**', allora c'è un elemento (o nessuno) all'interno di '**lista[a:z+1]**' e inoltre, '**lista[a:z+1]**' è già nel suo stato finale. Se '**z**' è maggiore di '**a**', allora (per assunzione) '**part**' ripartisce correttamente '**lista[a:z+1]**'. L'ordinamento separato dei due segmenti (per assunzione eseguito correttamente dalle chiamate ricorsive) completa l'ordinamento di '**lista[a:z+1]**'.

Le figure 3.31 e 3.32 mostrano due fasi della scansione effettuata da part all'interno dell'array o del segmento che gli viene fornito.

Figura 3.31. La scansione dell'array da parte di '**part**' avviene portando in avanti l'indice '**i**' e portando indietro l'indice '**cf**'. Quando l'indice '**i**' localizza un elemento che contiene un valore maggiore di '**lista[a]**' e l'indice '**cf**' localizza un elemento che contiene un valore inferiore o uguale a '**lista[a]**', gli elementi cui questi indici fanno riferimento vengono scambiati, quindi il processo di avvicinamento tra '**i**' e '**cf**' continua.

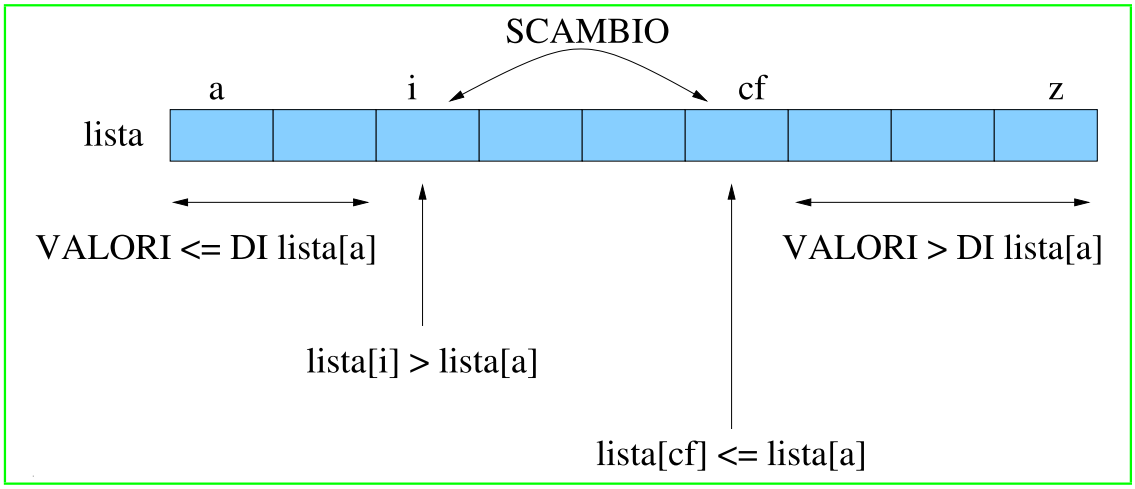
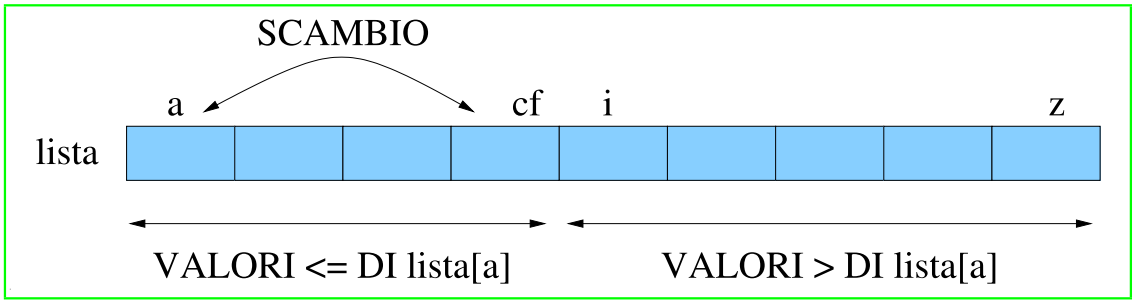


Figura 3.32. Quando la scansione è giunta al termine, quello che resta da fare è scambiare l'elemento '**lista[a]**' con '**lista[cf]**'.



In pratica, l'indice '**i**', iniziando dal valore '**a+1**', viene spostato verso destra fino a che viene trovato un elemento maggiore di '**lista[a]**', quindi è l'indice '**cf**' a essere spostato verso sinistra, iniziando dalla stessa posizione di '**z**', fino a che viene incontrato un elemento minore o uguale a '**lista[a]**'. Questi elementi vengono scambiati e lo spostamento di '**i**' e '**cf**' riprende. Ciò prosegue fino a che '**i**' e '**cf**' si incontrano, momento in cui '**lista[a:z+1]**' è stata ripartita e '**cf**' rappresenta la collocazione finale per l'elemento '**lista[l]**'.

La tabella 3.33 riassume la descrizione delle variabili utilizzate.

Tabella 3.33. Quicksort: variabili utilizzate.

Variabile	Descrizione
`lista`	L'array da ordinare in modo crescente.
`a`	L'indice inferiore del segmento di array da ordinare.
`z`	L'indice superiore del segmento di array da ordinare.
`cf`	Sta per «collocazione finale» ed è l'indice che cerca e trova la posizione giusta di '**lista[l]**' nell'array.
`i`	È l'indice che insieme a '**cf**' serve a ripartire l'array.

Il listato 3.34 presenta il programma Python che include le due funzioni.

Listato 3.34. Quicksort.

```
#!/usr/bin/python
##
## qsort.py <valore>...
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# part(<lista>, <inizio>, <fine>)
#
def part(lista, a, z):
  #
  # Viene preparata una variabile che serve per scambiare due valori.
  #
  scambio = 0
  #
  # Si assume che a sia inferiore a z.
```

```
  #
  i = a+1
  cf = z
  #
  # Inizia il ciclo di scansione dell'array.
  #
  while True:
    while True:
      #
      # Sposta i a destra.
      #
      if lista[i] > lista[a] or i >= cf:
        break
      else:
        i += 1
    while True:
      #
      # Sposta cf a sinistra.
      #
      if lista[cf] <= lista[a]:
        break
      else:
        cf -= 1
    if cf <= i:
      #
      # E' avvenuto l'incontro tra i e cf.
      #
      break
    else:
      #
      # Vengono scambiati i valori.
      #
      scambio = lista[cf]
      lista[cf] = lista[i]
      lista[i] = scambio
      i += 1
      cf -= 1
  #
  # A questo punto lista[a:z+1] e' stata ripartita e cf e' la
  # collocazione di lista[a].
  #
  scambio = lista[cf]
  lista[cf] = lista[a]
  lista[a] = scambio
  #
  # A questo punto, lista[cf] e' un elemento (un valore) nella
  # giusta posizione.
  #
  return cf
#
# quicksort(<lista>, <inizio>, <fine>)
#
def quicksort(lista, a, z):
```

```
  #
  # Viene preparata la variabile cf.
  #
  cf = 0
  #
  if z > a:
    cf = part(lista, a, z)
    quicksort(lista, a, cf-1)
    quicksort(lista, cf+1, z)
##
## Inizio del programma.
##
lista = sys.argv[1:]
#
quicksort(lista, 0, len(lista)-1);
#
for elemento in lista:
  print elemento,
#
```

Vale la pena di osservare che l'array viene indicato nelle chiamate in modo che alla funzione sia inviato un riferimento a quello originale, perché le variazioni fatte all'interno delle funzioni devono riflettersi sull'array originale.

3.3.3.1 Alcune osservazioni aggiuntive

In Python, non è necessario alcun particolare accorgimento sintattico per garantire questo comportamento: infatti, le liste costituiscono un tipo di dato ***mutabile*** (secondo la terminologia Python), alla stessa stregua di altri tipi come i ***dizionari*** (per i quali si rinvia il lettore alla documentazione del linguaggio[1]); quando un oggetto mutabile viene passato come argomento a una funzione, avviene un assegnamento al corrispondente parametro formale; l'assegnamento di un oggetto mutabile a una variabile è realizzato in Python mediante il cosiddetto meccanismo dell'***aliasing***: in pratica la nuova variabile coincide in tutto e per tutto con l'oggetto assegnato[2], e in particolare se quest'ultimo cambia valore tale cambiamento si riflette sulla nuova variabile. Pertanto, le variazioni fatte sui parametri formali all'interno di funzioni che ricevono come argomenti delle liste, si riflettono sulle liste originali.

Ecco un esempio che può aiutare a chiarire la questione (si tratta di una sessione interattiva dell'interprete Python):

```
$ python [ Invio ]

  Python 2.3.4 (#2, Jul  5 2004, 09:15:05)
  [GCC 3.3.4 (Debian 1:3.3.4-2)] on linux2
  Type "help", "copyright", "credits" or "license" for more information.


>>> lista = [1, 3, 7, 0, 10] [ Invio ]

>>> altra_lista = lista [ Invio ]

>>> altra_lista[3] = 1000 [ Invio ]
```

```
>>> print lista [ Invio ]

  [1, 3, 7, 1000, 10]


>>> [ Ctrl d ]

$
```

3.3.4 Permutazioni

La permutazione è lo scambio di un gruppo di elementi posti in sequenza. Il problema che si vuole analizzare è la ricerca di tutte le permutazioni possibili di un dato gruppo di elementi.

Se ci sono ***n*** elementi in un array, allora alcune delle permutazioni si possono ottenere bloccando l'***n***-esimo elemento e generando tutte le permutazioni dei primi elementi. Quindi l'***n***-esimo elemento può essere scambiato con uno dei primi ***n***−1, ripetendo poi la fase precedente. Questa operazione deve essere ripetuta finché ognuno degli ***n*** elementi originali è stato usato nell'***n***-esima posizione.

Tabella 3.37. Permutazioni: variabili utilizzate.

Variabile	Descrizione
`lista`	L'array da permutare.
`a`	L'indice inferiore del segmento di array da permutare.
`z`	L'indice superiore del segmento di array da permutare.
`k`	È l'indice che serve a scambiare gli elementi.

Il listato 3.38 presenta il programma Python, le cui variabili più importanti sono descritte nella tabella 3.37.

Listato 3.38. Permutazioni.

```
#!/usr/bin/python
##
## permuta.py <valore>...
##
#
# Importa il modulo sys, per usare sys.argv
#
import sys
#
# permuta(<lista>, <inizio>, <fine>)
#
def permuta(lista, a, z):
  #
  # Se il segmento di array contiene almeno due elementi, si
  # procede.
  #
  if z-a >= 1:
    #
```

```
    # Inizia un ciclo di scambi tra l'ultimo elemento e uno degli
    # altri contenuti nel segmento di array.
    #
    for k in range (z, a-1, -1):
      #
      # Scambia i valori.
      #
      scambio = lista[k]
      lista[k] = lista[z]
      lista[z] = scambio
      #
      # Esegue una chiamata ricorsiva per permutare un segmento
      # piu' piccolo dell'array.
      #
      permuta(lista, a, z-1)
      #
      # Scambia i valori.
      #
      scambio = lista[k]
      lista[k] = lista[z]
      lista[z] = scambio
  else:
    #
    # Visualizza la situazione attuale dell'array.
    #
    for elemento in lista:
      print elemento,
    print
##
## Inizio del programma.
##
lista = sys.argv[1:]
#
permuta(lista, 0, len(lista)-1)
#
```

3.3.4.1 Alcune osservazioni aggiuntive

Si colga l'occasione per notare un paio di aspetti idiomatici di Python. Per prima cosa, è possibile usare la funzione '**range**' per costruire un ciclo enumerativo decrescente, poiché '**range**' accetta un terzo argomento che in pratica rappresenta il passo con cui viene generata la successione di valori che popola la lista; ecco alcuni esempi:

```
$ python [ Invio ]

  Python 2.3.4 (#2, Jul  5 2004, 09:15:05)
  [GCC 3.3.4 (Debian 1:3.3.4-2)] on linux2
  Type "help", "copyright", "credits" or "license" for more information.


>>> range(10) [ Invio ]
```

```
  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(1, 11) [ Invio ]

  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> range(0, 30, 5) [ Invio ]

  [0, 5, 10, 15, 20, 25]

>>> range(0, 10, 3) [ Invio ]

  [0, 3, 6, 9]

>>> range(0, -10, -1) [ Invio ]

  [0, -1, -2, -3, -4, -5, -6, -7, -8, -9]

>>> range(0) [ Invio ]

  []

>>> range(1, 0) [ Invio ]

  []

>>> [ Ctrl d ]

$
```

si noti in particolare che, al solito, il secondo argomento denota il primo valore escluso dalla successione. Si noti infine l'uso dell'istruzione '**`print`**' isolata, allo scopo di andare a capo (ciò è necessario poiché nel ciclo enumerativo precedente si era usato il carattere vigola ('**`,`**') in coda, il quale sopprime il carattere di *newline*).

3.4 Un programma interattivo: «numeri.py»

In questa sezione viene presentato e commentato un semplice programma interattivo, '**`numeri.py`**', con lo scopo di illustrare alcune ulteriori caratteristiche (sintattiche e idiomatiche) del linguaggio Python.

3.4.1 Una sessione d'esempio

Prima del sorgente, è interessante vedere un esempio di esecuzione del programma, allo scopo di comprenderne meglio il funzionamento. Trattandosi di un programma interattivo, la sessione presentata dovrebbe commentarsi da sé.

```
$ ./numeri.py [ Invio ]

  ********
   Numeri
  ********

  Con questo programma e' possibile effettuare alcune
  operazioni matematiche su operandi scelti dall'utente.


Scegliere su quanti operandi operare [2-10]: pippo [ Invio ]

  Non e' un numero intero. Riprova...


Scegliere su quanti operandi operare [2-10]: 1 [ Invio ]

  Il numero dev'essere compreso fra 2 e 10. Riprova...


Scegliere su quanti operandi operare [2-10]: 3 [ Invio ]

  L'utente ha scelto di operare su 3 operandi.
  Inserire gli operandi su cui operare:


Operando 0: -7.9 [ Invio ]

Operando 1: pippo [ Invio ]

  Non e' un numero. Riprova...


Operando 1: 0.25 [ Invio ]

Operando 2: 123456 [ Invio ]

  L'utente ha scelto di operare sui seguenti operandi: -7.9, 0.25, 123456.0
  Operazioni consentite:
  0 - Termina
  1 - Addizione
  2 - Moltiplicazione
  3 - Massimo
  4 - Minimo
  5 - Media


Scegliere un'operazione [1-5, 0 per terminare]: 2 [ Invio ]
```

```
Scegliere un'operazione [1-5, 0 per terminare]: 1 [ Invio ]

Scegliere un'operazione [1-5, 0 per terminare]: 9 [ Invio ]

  Il numero dev'essere compreso fra 0 e 5. Riprova...


Scegliere un'operazione [1-5, 0 per terminare]: 5 [ Invio ]

Scegliere un'operazione [1-5, 0 per terminare]: 0 [ Invio ]

  Il prodotto degli operandi e' -243825.6
  La somma degli operandi e' 123448.35
  La media degli operandi e' 61726.0875


Desideri ricominciare da capo?  [s/N]: n [ Invio ]

  Grazie per aver utilizzato il programma numeri.py!


$
```

3.4.2 Il codice sorgente

Il listato 3.56 presenta il codice sorgente del programma '**numeri.py**'.

Listato 3.56. '**numeri.py**', un programma interattivo in Python.

```
1   #!/usr/bin/python
2   #=============================================================================
3   # Copyright (C) 2005 Massimo Piai <pxam67 (at) virgilio (dot) it>
4   #=============================================================================
5   ###
6   ### numeri.py
7   ###
8   ##
9   ## Importazione moduli essenziali
10  ##
11  import operator
12  import os
13  import sys
14  ##
15  ## Definizione delle varie funzioni
16  ##
17  #
18  # Calcola la media aritmetica
19  #
20  def med(operandi):
21    return reduce(operator.add,operandi)/len(operandi)
22  #
23  # Emette un'intestazione
24  #
25  def Intestazione():
```

```
26    print
27    print "********"
28    print " Numeri "
29    print "********"
30    print
31    print "Con questo programma e' possibile effettuare alcune"
32    print "operazioni matematiche su operandi scelti dall'utente."
33    print
34  #
35  # Elenca gli operandi
36  #
37  def Elenca_operandi():
38
39    for operando in operandi[:len(operandi)-1]:
40      print str(operando) + ",",
41
42    print operandi[len(operandi)-1]
43  #
44  # Elenca le operazioni possibili
45  #
46  def Elenca_operazioni_possibili():
47    print "Operazioni consentite:"
48
49    for i in range(len(nomi_operazioni)):
50      print i, "-", nomi_operazioni[i]
51  #
52  # Richiede all'utente l'indicazione del numero
53  # di operandi
54  #
55  def Richiesta_numero_operandi():
56    quanti_operandi = 0
57
58    while True:
59
60      try:
61        quanti_operandi = int(raw_input(
62          "Scegliere su quanti operandi operare [2-10]: "
63          ))
64      except ValueError:
65        print "Non e' un numero intero. Riprova..."
66      else:
67
68        if quanti_operandi in range(2,11):
69          break
70        else:
71          print "Il numero dev'essere compreso fra 2 e 10. Riprova..."
72
73    print
74    return quanti_operandi
75  #
76  # Richiede all'utente di inserire gli operandi
77  #
78  def Richiesta_inserimento_operandi():
```

```
79    print "L'utente ha scelto di operare su", quanti_operandi, "operandi."
80    print "Inserire gli operandi su cui operare:"
81    operandi = []
82
83    for i in range(quanti_operandi):
84
85      while True:
86
87        try:
88          operando = float(raw_input("Operando " + str(i) + ": "))
89          operandi.append(operando)
90          break
91        except ValueError:
92          print "Non e' un numero. Riprova..."
93
94    print
95    return operandi
96  #
97  # Richiede all'utente di scegliere le operazioni
98  #
99  def Richiesta_scelta_operazioni():
100   print "L'utente ha scelto di operare sui seguenti operandi:",
101   Elenca_operandi()
102   Elenca_operazioni_possibili()
103   operazioni = []
104
105   while True:
106
107     try:
108       opz_n = int(raw_input(
109         "Scegliere un'operazione [1-5, 0 per terminare]: "
110         ))
111     except ValueError:
112       print "Non e' un numero intero. Riprova..."
113     else:
114
115       if opz_n in range(len(operazioni_possibili)+1):
116
117         if not opz_n:
118           break
119         else:
120
121           if nomi_operazioni[opz_n] not in operazioni:
122             operazioni.append(nomi_operazioni[opz_n])
123           else:
124             print "Operazione gia' scelta."
125
126       else:
127         print "Il numero dev'essere compreso fra 0 e 5. Riprova..."
128
129   print
130   return operazioni
131 #
```

```
132 # Presenta i risultati e offre la possibilita‘ di
133 # ricominciare o terminare
134 #
135 def Calcolo_e_presentazione_risultati():
136
137   for operazione in operazioni:
138     print nomi_risultati_operazioni[operazione], "degli operandi e‘",
139
140     if operazione in operazioni_n_arie:
141       print operazioni_possibili[operazione](operandi)
142     else:
143       print reduce(operazioni_possibili[operazione], operandi)
144
145   print
146
147   return raw_input(
148     "Desideri ricominciare da capo? [s/N]: "
149     ).lower().startswith("s")
150 #
151 # Gestione ringraziamenti e saluti
152 #
153 def Commiato():
154   print "Grazie per aver utilizzato il programma",
155   print os.path.basename(sys.argv[0]) + "!"
156   print
157 ##
158 ## Inizio programma principale
159 ##
160 #
161 # Variabili globali
162 #
163 continua = True
164 operandi = []
165 operazioni = []
166 quanti_operandi = 0
167 operazioni_possibili = {"Addizione": operator.add,
168                         "Moltiplicazione": operator.mul,
169                         "Massimo": max,
170                         "Minimo": min,
171                         "Media": med}
172 nomi_risultati_operazioni = {"Addizione": "La somma",
173                              "Moltiplicazione": "Il prodotto",
174                              "Massimo": "Il massimo",
175                              "Minimo": "Il minimo",
176                              "Media": "La media"}
177 nomi_operazioni = ["Termina", "Addizione", "Moltiplicazione",
178                    "Massimo", "Minimo", "Media"]
179 operazioni_n_arie = ["Massimo", "Minimo", "Media"]
180 #
181 #
182 #
183 Intestazione()
184
```

```
185  while continua:
186    quanti_operandi = Richiesta_numero_operandi()
187    #
188    # L'utente ha scelto su quanti operandi operare,
189    # quindi si procede a chiederne l'inserimento
190    #
191    operandi = Richiesta_inserimento_operandi()
192    #
193    # L'utente ha inserito gli operandi, quindi
194    # si procede a chiedergli che operazione eseguire
195    #
196    operazioni = Richiesta_scelta_operazioni()
197    #
198    # L'utente ha scelto le operazioni,
199    # si procede quindi al calcolo e alla
200    # presentazione dei risultati, offrendo
201    # la possibilita' di ricominciare o terminare
202    #
203    continua = Calcolo_e_presentazione_risultati()
204    print
205  #
206  # Commiato dall'utente
207  #
208  Commiato()
```

3.4.3 Analisi e commento

Commentiamo gli aspetti principali del programma. Cominciamo l'analisi dal livello più esterno:

- **Righe 157-176**

  Vengono definite alcune variabili (o ***nomi*** secondo la terminologia Python) utilizzate dal programma principale.

  La variabile '**`operazioni_possibili`**' è un dizionario che serve a mantenere una corrispondenza fra stringhe (che contengono dei nomi convenzionali di funzioni) e funzioni (le quali possono essere incorporate oppure appartenenti a un modulo esterno oppure ancora definite altrove nel programma stesso). '**`nomi_risultati_operazioni`**' è un dizionario che mantiene una corrispondenza fra fra stringhe che contengono dei nomi convenzionali di funzioni e stringhe che contengono i nomi tradizionali che esprimono il risultato del calcolo delle funzioni stesse. '**`nomi_operazioni`**' è una lista che contiene i nomi delle funzioni che l'utente può di volta in volta scegliere di utilizzare. '**`operazioni_n_arie`**' è una lista che contiene i nomi delle funzioni che sono definite in modo da operare su 2 o più argomenti (le rimanenti funzioni operano esattamente su 2 argomenti).

- **Righe 177-205**

  È il programma principale. Si compone essenzialmente di un ciclo iterativo che si ripete fintantoché la variabile booleana '**`continua`**' risulta vera. La successione dei passi che compongono il corpo del ciclo è la seguente:

  – richiedere all'utente su quanti operandi operare, e assegnazione di tale valore a '**`quanti_operandi`**';

  - richiedere all'utente su quali operandi operare, e assegnazione di tali valori alla lista '**`operandi`**';
  - richiedere all'utente con quali operazioni operare, e assegnazione di tali valori alla lista '**`operazioni`**';
  - calcolo e presentazione dei risultati; richiesta all'utente se vuole continuare con una nuova iterazione e assegnazione corrispondente alla variabile '**`continua`**'.

  All'uscita del ciclo il programma termina dopo essersi accomiatato dall'utente.

I vari passi che costituiscono il ciclo principale sono realizzati mediante funzioni che restituiscono un valore adeguato alla necessità. Procediamo ad analizzare gli aspetti maggiormente qualificanti di tali funzioni.

- **Righe 31-39**

  La funzione '**`Elenca_operandi`**' elenca i membri della lista '**`operandi`**' separati da virgole.

  Si noti l'idioma tipico per iterare su di una lista fino al penultimo membro:

  ```
  for membro in lista[:len(lista)-1]:
    ...
  ```

  Ciò è necessario per trattare l'ultimo membro come caso speciale.

- **Righe 40-47**

  La funzione '**`Elenca_operazioni_possibili`**' emette un elenco numerato delle operazioni possibili per permettere all'utente la scelta.

  Si noti il tipico idioma per iterare su di una lista attraverso gli indici invece che attraverso i membri:

  ```
  for indice in range(len(lista)):
    ...
  ```

- **Righe 48-71**

  La funzione '**`Richiesta_numero_operandi()`**' riceve in input il numero di operandi su cui operare e lo restituisce al chiamante; effettua anche un controllo piuttosto dettagliato dell'input, tramite i costrutti Python per la gestione delle ***eccezioni***[3]: '**`try`**', '**`except`**' e '**`else`**'. In pratica la funzione esegue un ciclo infinito ('**`while True:`**') all'interno del quale riceve l'input e cerca ('**`try:`**') di convertirlo in un intero ('**`...int(raw_input(...`**'); se la conversione fallisce ('**`except ValueError:`**') il ciclo continua; se la conversione ha successo ma il valore non è consentito il ciclo continua; altrimenti il ciclo termina ('**`break`**'[4]).

- **Righe 72-92**

- La funzione '**`Richiesta_inserimento_operandi`**' costruisce la lista degli operandi ricevuti in input e la restituisce al chiamante; il controllo dell'input viene effettuato con la tecnica già vista della gestione delle eccezioni.

Si noti l'idioma tipico per l'estensione di una lista con un membro in coda:

```
lista.append(membro)
```

- **Righe 93-127**

  La funzione '**`Richiesta_scelta_operazioni`**' chiama a sua volta '**`Elenca_operandi`**' e '**`Elenca_operazioni_possibili`**', dopodiché prepara e restituisce al chiamante la lista delle operazioni (ossia delle stringhe contenenti i nomi delle operazioni come da dizionario '**`operazioni_possibili`**') richiesta dall'utente. L'utente sceglie le operazioni mediante il numero progressivo mostrato da '**`Elenca_operazioni_possibili`**' e l'input viene via via controllato mediante una tecnica simile a quelle già incontrate. Si noti inoltre:

  – la struttura condizionale '**`if not opz_n:`**' la quale ha successo esattamente quando '**`opz_n`**' vale 0[5]; in tal caso il ciclo termina;

  – l'istruzione di estensione della lista delle operazioni richieste è controllata dalla struttura condizionale '**`if nomi_operazioni[opz_n] not in operazioni:`**' per far sì che non venano inseriti doppioni nella lista.

  Si tratta di un idioma Python tipico per controllare la presenza di un membro in una lista:

  ```
  membro in lista
  ```

  oppure:

  ```
  membro not in lista
  ```

- **Righe 128-146**

  La funzione '**`Calcolo_e_presentazione_risultati`**' calcola le operazioni richieste sugli operandi indicati, presenta i risultati e chiede all'utente se desidera ricominciare. L'applicazione delle funzioni che realizzano le operazioni agli operandi viene effettuata in due modi diversi a seconda che la funzione sia ***binaria*** (ossia lavori esattamente su 2 argomenti) oppure ***ternaria*** o più: nel primo caso su utilizza la funzione Python '**`reduce`**' la quale per l'appunto estende le funzioni binarie al caso di più argomenti[6].

  '**`raw_input(...`**' chiede all'utente di inserire un'input; la stringa viene convertita in minuscole tramite il metodo '**`lower`**'; il metodo '**`startswith("s")`**' restituisce un valore booleano a seconda che la stringa cui viene applicato inizi o meno con '**`"s"`**'.

3.4.3.1 Alcune osservazioni aggiuntive

Le variabili di cui alle righe 157-176, essendo dichiarate nel blocco più esterno del sorgente, queste potrebbero essere chiamate «variabili globali», secondo una tradizione consolidata: in effetti tali nomi sono accessibili (in lettura) in tutto il programma, mentre l'accesso in scrittura è possibile solamente nel livello più esterno.

Più precisamente, in Python è possibile accedere - a un livello più interno - a un nome dichiarato al livello più esterno, anche in scrittura, ma:

- il valore associato al nome è quello del livello esterno fino alla modifica, e
- la modifica perde effetto appena il controllo ritorna al livello esterno.

È possibile alterare quest'ultimo comportamento utilizzando la parola chiave '**`global`**': dichiarando come ***global*** un nome, ogni modifica al livello interno si riflette al livello esterno. Ad esempio:

```
def funz():
  global x
  x = 0
  print "sono funz"
  print "x: ", x

x = 42
print "sono il programma principale"
print "x: ", x
funz()
print "sono il programma principale"
print "x: ", x
```

```
$ python tmp/global.py [ Invio ]

  sono il programma principale
  x:  42
  sono funz
  x:  0
  sono il programma principale
  x:  0


$
```

Senza la dichiarazione '**`global x`**' l'esecuzione avrebbe fornito il seguente output:

```
sono il programma principale
x:  42
sono funz
x:  0
sono il programma principale
x:  42
```

Per evitare ambiguità, nel seguito chiameremo nomi o variabili ***global*** i nomi Python dichiarati con la parola chiave '**`global`**'.

[1] Essenzialmente un dizionario è un tipo speciale di array: in pratica si tratta di una collezione di coppie chiave-valore, in cui le chiavi possono essere di qualsiasi tipo immutabile, e non solamente numeri interi; corrispondono in pratica agli array associativi del linguaggio Perl.

[2] In altri termini: argomento e parametro formale sono due nomi dello stesso oggetto Python.

[3] Trattasi di errori non sintattici non necessariamente fatali intercettati in fase di esecuzione.

[4] Si tenga presente che la parola chiave può comparire solo nel corpo di un ciclo (enumerativo oppure iterativo) e ha l'effetto di terminare immediatamente il ciclo più interno fra quelli che la includono.

[5] In Python sono considerati valori booleani falsi ('`False`'): '`None`', lo zero numerico di qualunque tipo, la stringa vuota ('`""`'), la tupla vuota ('`()`'), la lista vuota ('`[]`'), il dizionario vuoto ('`{}`'). Tutti gli altri valori sono interpretati come veri ('`True`').

[6] '`reduce`' viene utilizzata anche nelle righe 11-18 ove viene definita la funzione '`med`' la quale calcola la media artitmetica dei membri della lista che le viene passata come argomento.

Parte ii

Alml per sopravvivere

In questa parte si parla di Alml, il sistema di composizione SGML realizzato da Daniele Giacomini per la gestione dei suoi *Appunti di informatica libera*.

Alml: estensioni PXAM

A partire dall'estate del 2005, Alml permette di incorporare codice relativo a svariati tipi di minilinguaggi grafici.

4.1 Immagini incorporate macro *roff: Pic, Tbl, Eqn

È possibile incorporare codice relativo ai pacchetti di macro *roff denominati Pic, Tbl, Eqn attraverso gli elementi '**picimg**', '**tblimg**', '**eqnimg**' rispettivamente; l'utilizzo è abbastanza analogo a quelli di altri elementi simili.

Nei listati 4.1, 4.2, 4.3 vengono presentati tre esempi di incorporazione nell'ambito degli elementi '**picimg**', '**tblimg**', '**eqnimg**' rispettivamente; corrispondentemente, le figure 4.4, 4.5, 4.6 illustrano l'aspetto finale delle immagini in relazione al codice presentato.

Listato 4.1. Codice incorporato: macro *roff (Pic).

```
...

<object>
<imgblock>
<picimg>
<![CDATA[
ellipse "document";
arrow;
box width 0.6 "\fIpic\/\fP(1)"
arrow;
box width 1.1 "\fIgtbl\/\fP(1) or \fIgeqn\/\fP(1)" "(optional)" dashed;
arrow;
box width 0.6 "\fIgtroff\/\fP(1)";
arrow;
ellipse "PostScript"
]]>
</picimg>
</imgblock>
</object>

...
```

Listato 4.2. Codice incorporato: macro *roff (Tbl).

```
...

<object>
<imgblock>
<tblimg>
<![CDATA[
tab(:);
c s s
c | c | c
l | l | n.
Major New York Bridges
=
Bridge:Designer:Length
_
Brooklyn:J. A. Roebling:1595
Manhattan:G. Lindenthal:1470
Williamsburg:L. L. Buck:1600
_
Queensborough:Palmer &:1182
\^:  Hornbostel:\^
_
Triborough:O. H. Ammann:1380
\^:\^:383
_
Bronx Whitestone:O. H. Ammann:2300
Throgs Neck:O. H. Ammann:1800
_
George Washington:O. H. Ammann:3500
_
]]>
</tblimg>
</imgblock>
</object>

...
```

Listato 4.3. Codice incorporato: macro *roff (Eqn).

```
...

<object>
<imgblock>
<eqnimg>
<![CDATA[
G(z)
~=~ e sup { ln ~ G(z) }
~=~ exp left ( sum from k>=1 { S sub k z sup k } over k right )
~=~ prod from k>=1 e sup { S sub k z sup k /k }
]]>
</eqnimg>
</imgblock>
```

```
</object>

...
```

Figura 4.4. Codice incorporato: Pic.

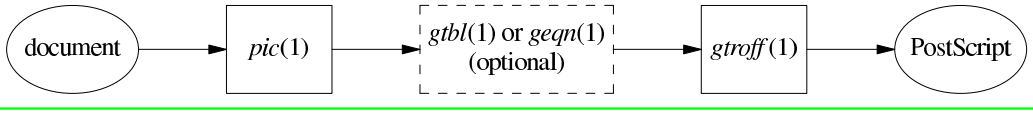


Figura 4.5. Codice incorporato: Tbl.

Major New York Bridges

Bridge	Designer	Length
Brooklyn	J. A. Roebling	1595
Manhattan	G. Lindenthal	1470
Williamsburg	L. L. Buck	1600
Queensborough	Palmer & Hornbostel	1182
Triborough	O. H. Ammann	1380 383
Bronx Whitestone	O. H. Ammann	2300
Throgs Neck	O. H. Ammann	1800
George Washington	O. H. Ammann	3500

Figura 4.6. Codice incorporato: Eqn.

$$G(z) = e^{\ln G(z)} = \exp\left(\sum_{k\geq 1} \frac{S_k z^k}{k}\right) = \prod_{k\geq 1} e^{S_k z^k/k}$$

4.1.1 Alcune osservazioni aggiuntive

Si notino le seguenti particolarità:

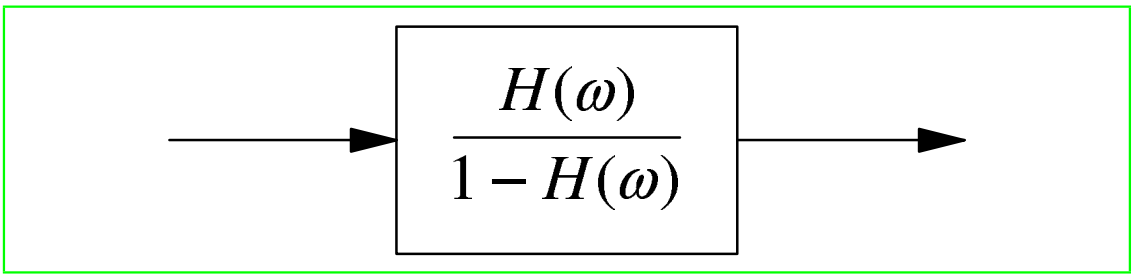
- **non** si deve delimitare il codice mediante le macro '**.PS**' e '**.PE**', '**.TS**' e '**.TE**', '**.EQ**' e '**.EN**' rispettivamente, sebbene esse siano previsti in un sorgente *roff normale;
- il codice Pic e il codice Tbl possono includere codice Eqn, a patto di delimitare tale codice mediante una coppia di caratteri dollaro ('**$**') (il listato 4.7 e la figura 4.8 illustrano un esempio nel primo caso).

Listato 4.7. Codice incorporato: Eqn dentro Pic.

```
...

<object>
<imgblock>
<picimg>
<![CDATA[
arrow
box "$space 0 {H( omega )} over {1 - H( omega )}$"
```

```
arrow
]]>
</picimg>
</imgblock>
</object>

...
```

Figura 4.8. Codice incorporato: Eqn dentro Pic.



4.2 Immagini incorporate GNU Plotutils: Graph, Plot, Pic2plot, Spline, Ode

È possibile incorporare codice relativo ai programmi del pacchetto GNU Plotutils,[1] Graph, Plot, Pic2plot, Spline, Ode attraverso gli elementi '**pugraphimg**', '**puplotimg**', '**pupic2plotimg**', '**pusplineimg**' e '**puodeimg**' rispettivamente; l'utilizzo è abbastanza analogo a quelli di altri elementi simili.

Nei listati 4.9, 4.10, 4.11, 4.12, 4.13 vengono presentati cinque esempi di incorporazione nell'ambito degli elementi '**pugraphimg**', '**puplotimg**', '**pupic2plotimg**', '**pusplineimg**', '**puodeimg**' rispettivamente; corrispondentemente, le figure 4.14, 4.15, 4.16, 4.17, 4.18 illustrano l'aspetto finale delle immagini in relazione al codice presentato.

Listato 4.9. Codice incorporato: GNU Plotutils (Graph).

```
...

<object>
<imgblock>
<pugraphimg>
<![CDATA[
#-h .3 -w .6
0.0   0.0
1.0   0.2
2.0   0.0
3.0   0.4
4.0   0.2
5.0   0.6
6.0   0.2
7.0   0.0
8.0   0.4
9.0   0.2
10.0  0.6
]]>
```

```
</pugraphimg>
</imgblock>
</object>

...
```

Listato 4.10. Codice incorporato: GNU Plotutils (Plot).

```
...

<object>
<imgblock>
<puplotimg>
<![CDATA[
#PLOT 2
o
W 0.003
m 0 0.3
t rettangoli:
M -0.1 -0.1
H 0 0 0.01 0.08
M 0.05 0
H 0 0 0.02 0.04
M 0.05 0
H 0 0 0.04 0.02
M 0.05 0
H 0 0 0.08 0.01
#
M -0.215 -0.1
t segmenti:
M -0.1 -0.1
I 0 0 0.01 0.08
M 0.05 -0.08
I 0 0 0.02 0.04
M 0.05 -0.04
I 0 0 0.04 0.02
M 0.05 -0.02
I 0 0 0.08 0.01
#
M -0.29 -0.05
t ellissi:
M -0.05 -0.05
= 0 0 0.02 0.04 0
M 0.08 0
= 0 0 0.02 0.04 20
M 0.08 0
= 0 0 0.02 0.04 40
M 0.08 0
= 0 0 0.02 0.04 60
#
M -0.25 -0.07
t coniche di b\'ezier:
M -0.15 -0.07
```

```
r 0 0 0.05 0.03 0 0.06
M 0.08 -0.06
r 0 0 0.06 0.04 0 0.06
M 0.08 -0.06
r 0 0 0.07 0.05 0 0.06
M 0.08 -0.06
r 0 0 0.08 0.06 0 0.06
x
]]>
</puplotimg>
</imgblock>
</object>

...
```

Listato 4.11. Codice incorporato: GNU Plotutils (Pic2plot).

```
...

<object>
<imgblock>
<pupic2plotimg>
<![CDATA[
.PS
box "START"; arrow; circle dashed filled; arrow
circle diam 2 thickness 3 "This is a" "big, thick" "circle" dashed; up
arrow from top of last circle; ellipse "loopback" dashed
arrow dotted from left of last ellipse to top of last box
arc cw radius 1/2 from top of last ellipse; arrow
box "END"
.PE
]]>
</pupic2plotimg>
</imgblock>
</object>

...
```

Listato 4.12. Codice incorporato: GNU Plotutils (Spline).

```
...

<object>
<imgblock>
<pusplineimg>
<![CDATA[
#-d 2 -a -s -p -T -14 -n 500
#
0 0 1 0 1 1 0 0
]]>
</pusplineimg>
</imgblock>
```

```
</object>

...
```

Listato 4.13. Codice incorporato: GNU Plotutils (Ode).

```
...

<object>
<imgblock>
<puodeimg>
<![CDATA[
#
#-C -x 0 5 -y 0 5
x' = (a - b*y) * x
y' = (c*x - d) * y
a = 1; b = 1; c = 1; d = 1;
print x, y
x = 1; y = 2
step 0, 10
x = 1; y = 3
step 0, 10
x = 1; y = 4
step 0, 10
x = 1; y = 5
step 0, 10
]]>
</puodeimg>
</imgblock>
</object>

...
```

Figura 4.14. Codice incorporato: Graph.

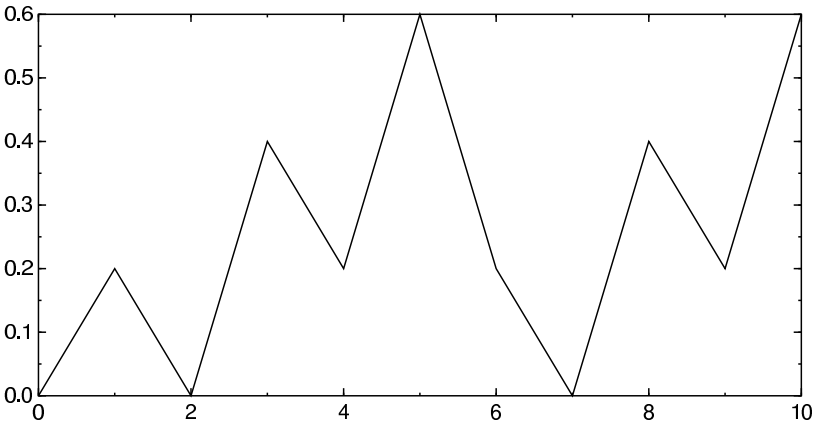


Figura 4.15. Codice incorporato: Plot.

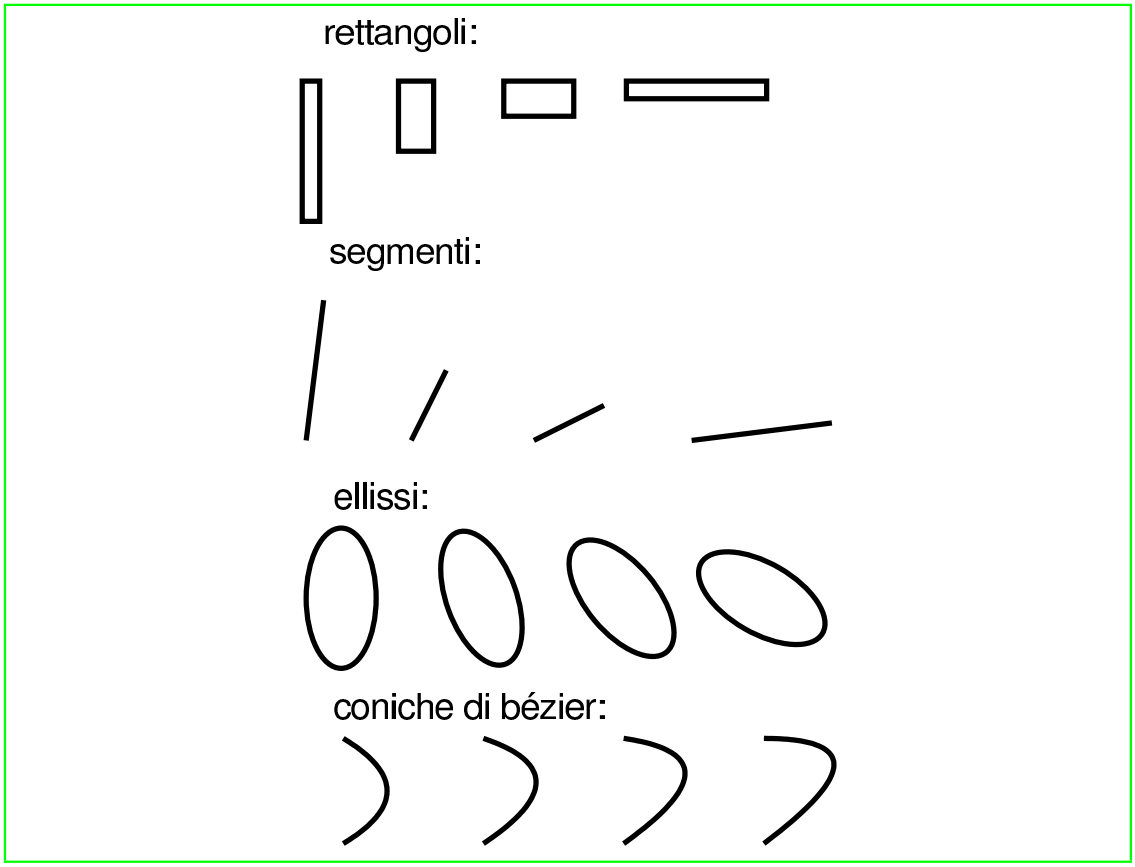


Figura 4.16. Codice incorporato: Pic2plot.

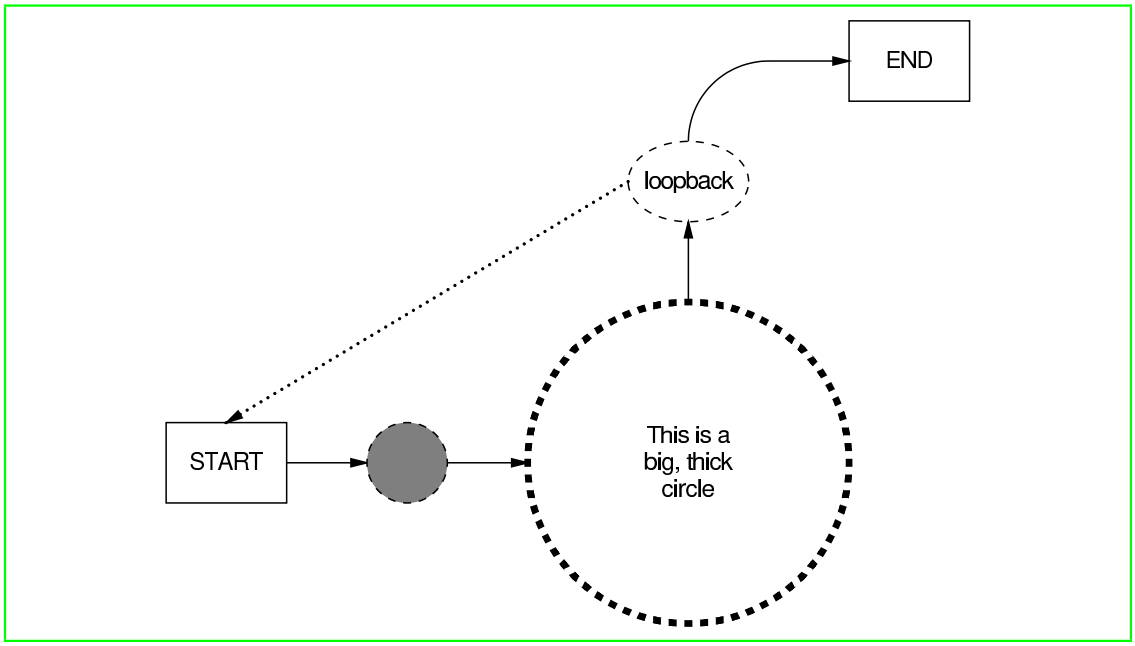


Figura 4.17. Codice incorporato: Spline.

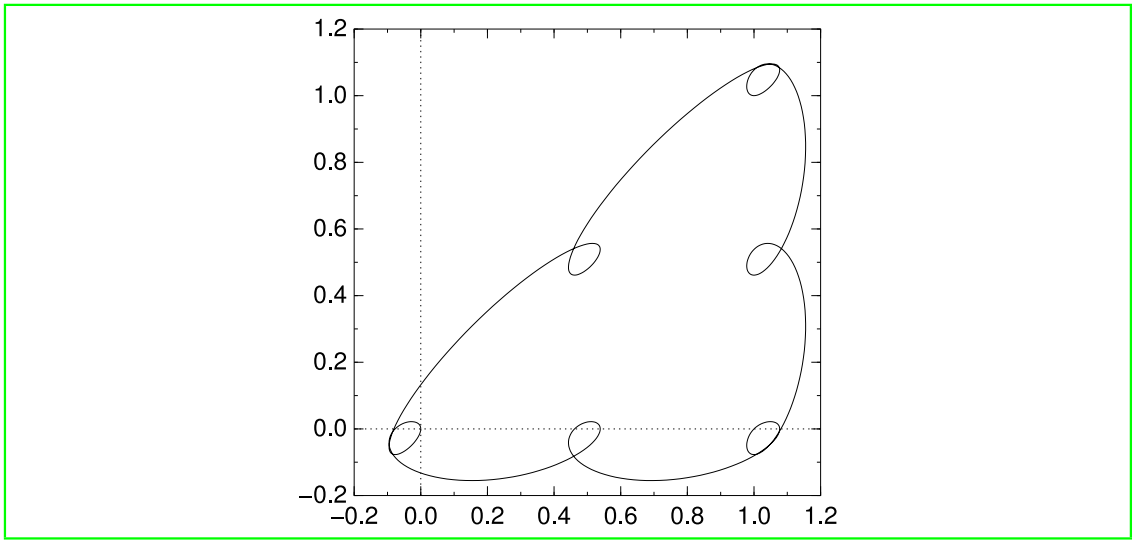
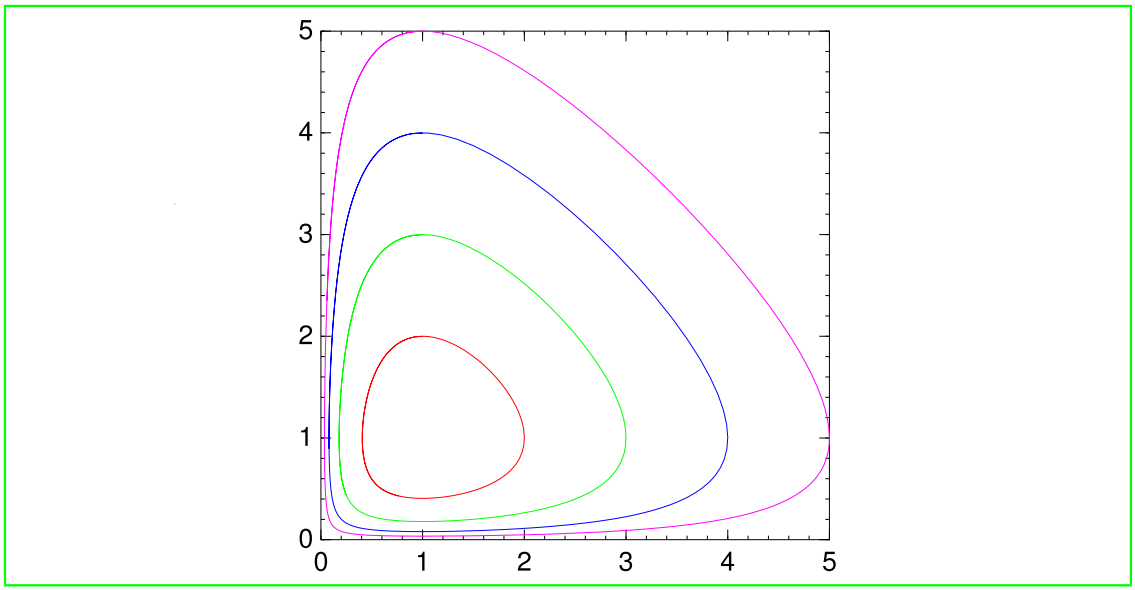


Figura 4.18. Codice incorporato: Ode.



4.2.1 Alcune osservazioni aggiuntive

Si notino le seguenti particolarità:

- la prima riga del codice Graph **deve** essere un commento (eventualmente vuoto), ossia una riga che inizi con il carattere cancelletto (‘`#`’); quanto segue il primo carattere viene interpretato come riga di comando da passare all’eseguibile ‘**`graph`**’, al fine di consentire l’uso delle varie opzioni del programma;
- quanto detto al punto precedente si applica anche al caso del codice Spline e del codice Ode, con la differenza che in questi casi sono **due** le righe di commento obbligatorie, corrispondenti a righe di comando per gli eseguibili ‘**`spline`**’ e ‘**`graph`**’, oppure ‘**`ode`**’ e ‘**`graph`**’, rispettivamente (poiché entrambe le coppie di eseguibili contribuiscono, in una pipeline, alla generazione dell’immagine);
- a causa del meccanismo con cui ‘**`alml`**’ estrae le righe di comando dalla sezione marcata ‘**`CDATA`**’, è **essenziale** che non ci siano spazi o righe vuote fra il marcatore precedente e

l'inizio della sezione stessa; ad esempio, la riga 3 nel seguente listato **deve** terminare con il carattere '**>**':

```
1  ...
2  <puodeimg>
3  <![CDATA[
4  #
5  #-C -x 0 5 -y 0 5
6  ...
```

- il codice Pic2plot **deve** iniziare con la macro '**.PS**' e terminare con la macro '**.PE**'; inoltre si tenga presente che l'elemento '**picimg**' (sezione 4.1) fornisce un supporto più completo per chi desideri utilizzare codice Pic (ad esempio consente l'uso dei colori, nonché l'inclusione di codice Eqn).

4.3 Immagini incorporate: codice generico

È possibile incorporare codice grafico generico, nel senso di una qualunque riga di comando associata a codice in input, a patto che tale riga di comando sia opportunamente formattata e il comando emetta codice EPS; si utilizza a tal fine l'elemento '**genericimg**'.

Nei listati 4.20 e 4.21 vengono presentati due esempi di incorporazione nell'ambito dell'elemento '**genericimg**'; corrispondentemente, le figure 4.22 e 4.23 illustrano l'aspetto finale delle immagini in relazione al codice presentato.

Listato 4.20. Codice incorporato generico.

```
...

<object>
<imgblock>
<genericimg>
<![CDATA[
#convert -resize 3000% -negate -charcoal 0 -swirl 60 PPM:INPUT EPS:OUTPUT
P3
# feep.ppm
4 4
15
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
]]>
</genericimg>
</imgblock>
</object>

...
```

Listato 4.21. Codice incorporato generico: un caso limite.

```
...

<object>
<imgblock>
<genericimg>
<![CDATA[
#echo "set terminal table; set samples 1000; plot [-20:30] 0.1*x**2*cos(0.3*x), ↩
↪20*sin(x)" | gnuplot | graph -I g -T ps > OUTPUT
#
# Questo è un caso limite: in pratica non c'è input per il comando
# presente nella prima riga: fa tutto lui!
#
]]>
</genericimg>
</imgblock>
</object>

...
```

Figura 4.22. Codice incorporato generico.

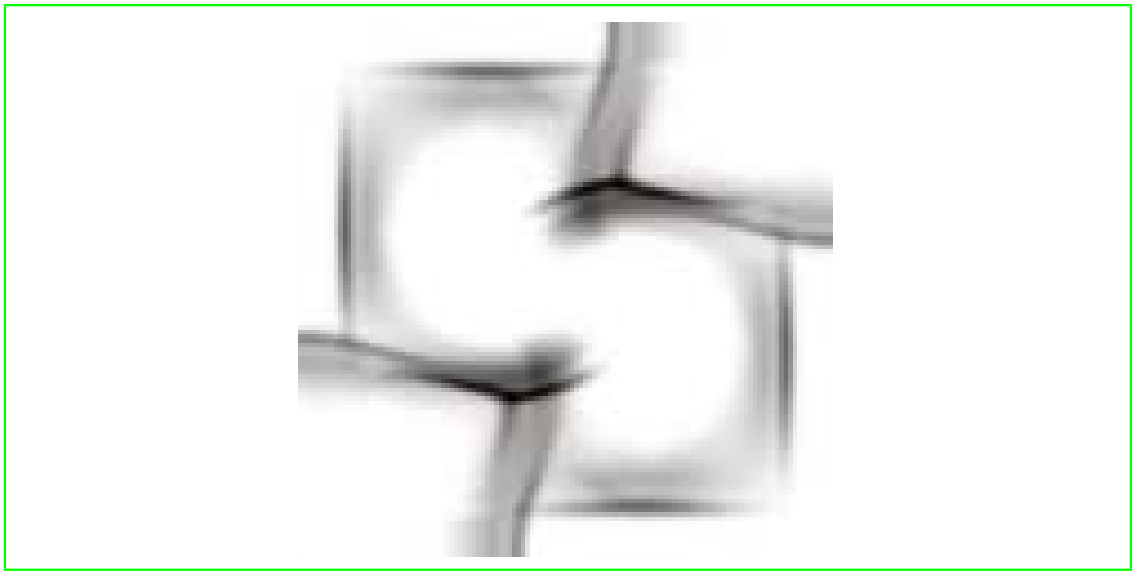
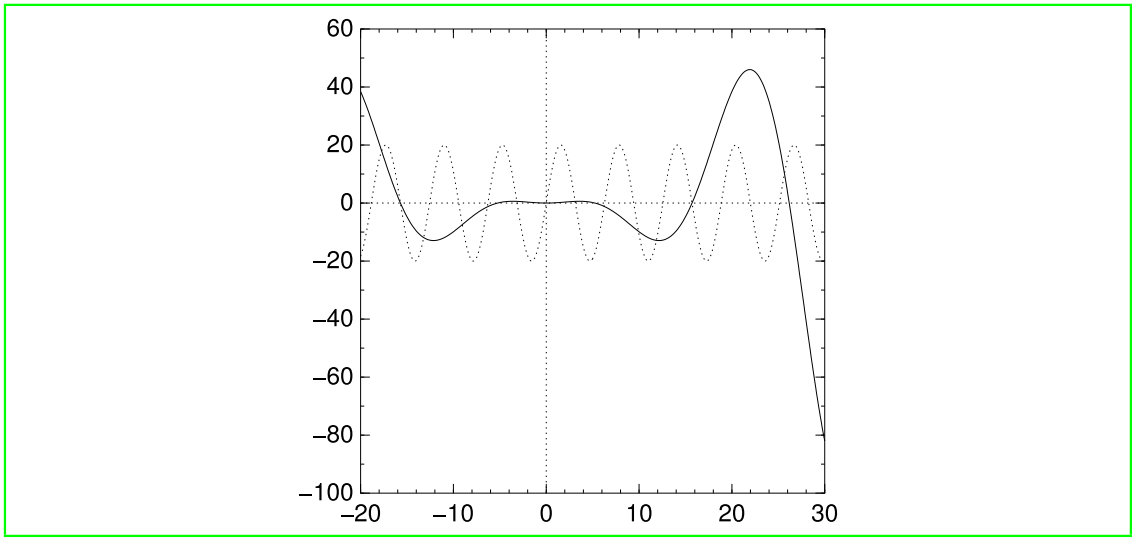


Figura 4.23. Codice incorporato generico: un caso limite.



4.3.1 Alcune osservazioni aggiuntive

Si notino le seguenti particolarità:

- la **prima riga** del codice ha la forma di un commento standard; quanto segue il carattere cancelletto (‘`#`’) è lo schema del comando da eseguire: in esso possono figurare (in guisa di metavariabili) le stringhe ‘**`INPUT`**’ e ‘**`OUTPUT`**’, le quali rappresentano rispettivamente l’input e l’output della riga di comando; l’input viene in pratica prelevato dalle rimanenti righe del codice, mentre l’output deve essere in formato EPS;
- le righe del codice successive alla prima **devono** costituire un input valido per la riga di comando, ma eventualmente **possono** essere assenti, nel caso in cui la riga di comando non necessiti di input (listato 4.20 e figura 4.23);
- valgono anche nel caso del codice generico le osservazioni relative agli spazi o alle righe vuote immediatamente prima dell’inizio della sezione marcata ‘**`CDATA`**’, come nel caso del codice GNU Plotutils Graph, Spline e Ode;
- per ragioni di sicurezza la riga di comando **non** viene eseguita (e l’immagine, di conseguenza, non viene generata), a meno che l’eseguibile ‘**`alml`**’ non venga invocato utilizzando l’opzione ‘**`--embedded-script-enable`**’.

4.4 Diagrammi scacchistici

È possibile incorporare codice per tracciare diagrammi e annotazioni scacchistiche di vario genere attraverso gli elementi ‘**`fenimg`**’, ‘**`fenitimg`**’, ‘**`sanimg`**’, ‘**`sanitimg`**’ e ‘**`skakimg`**’.

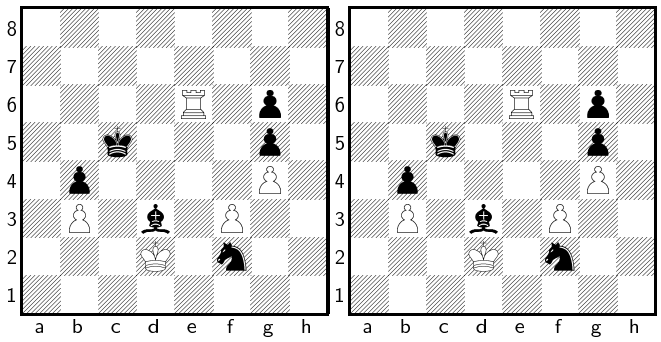
L’elemento ‘**`fenimg`**’ permette di definire una posizione sulla scacchiera mediante la notazione Forsyth-Edwards (FEN ovvero *Forsyth-Edwards Notation*). La notazione FEN permette di indicare i pezzi mediante singole lettere che in pratica costituiscono l’iniziale del nome del pezzo stesso nella lingua inglese (‘**`K`**’ significa «King» ossia il re bianco, ‘**`k`**’ significa «king» ossia il re nero, ecc.); volendo utilizzare le iniziali dei nomi nella lingua italiana è necessario utilizzare l’elemento ‘**`fenitimg`**’.

Nel listato 4.24 vengono presentati due esempi di incorporazione nell'ambito degli elementi '**fenimg**' e '**fenitimg**' rispettivamente; corrispondentemente, la figura 4.25 illustra l'aspetto finale delle immagini in relazione al codice presentato.

Listato 4.24. Codice incorporato FEN.

```
...

<object>
<imgblock>
<fenimg>
<![CDATA[
8/8/4R1p1/2k3p1/1p4P1/1P1b1P2/3K1n2/8 b - - 0 43
]]>
</fenimg>
<fenitimg>
<![CDATA[
8/8/4T1p1/2r3p1/1p4P1/1P1n1P2/3R1c2/8 n - - 0 43
]]>
</fenitimg>
</imgblock>
</object>

...
```

Figura 4.25. Codice incorporato FEN: nel primo diagramma i pezzi sono stati indicati secondo la denominazione in lingua inglese, nel secondo diagramma secondo la denominazione italiana.



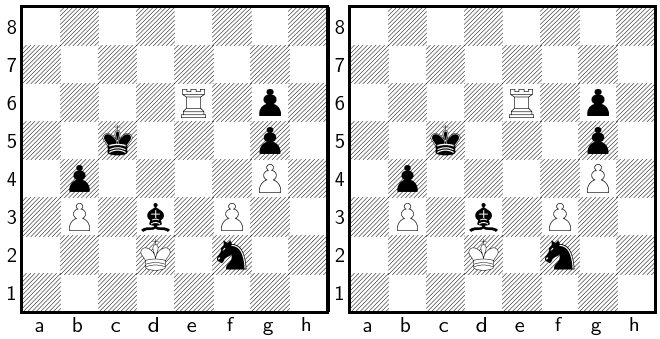
L'elemento '**sanimg**' permette di definire una posizione sulla scacchiera **come risultato** di una serie di mosse a partire dalla posizione iniziale, espresse mediante la notazione algebrica standard (SAN ovvero *Standard Algebraic Notation*). Nella notazione SAN i pezzi vengono indicati mediante singole lettere che in pratica costituiscono l'iniziale del nome del pezzo stesso nella lingua inglese; volendo utilizzare le iniziali dei nomi nella lingua italiana è necessario utilizzare l'elemento '**sanitimg**'.

Nel listato 4.26 vengono presentati due esempi di incorporazione nell'ambito degli elementi '**sanimg**' e '**sanitimg**' rispettivamente; corrispondentemente, la figura 4.27 illustra l'aspetto finale delle immagini in relazione al codice presentato.

Listato 4.26. Codice incorporato SAN: partita svoltasi a Belgrado, il 4 novembre 1992, fra Bobby Fischer e Boris Spassky.

```
...

<object>
<imgblock>
<sanimg>
<![CDATA[
1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5
7. Bb3 d6 8. c3 O-O 9. h3 Nb8 10. d4 Nbd7 11. c4 c6 12. cxb5
axb5 13. Nc3 Bb7 14. Bg5 b4 15.  Nb1 h6 16. Bh4 c5 17. dxe5
Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21.  Nc4 Nxc4
22. Bxc4 Nb6 23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1
Kxf7 27. Qe3 Qg5 28. Qxg5 hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4
cxb4 32. Ra5 Nd5 33.  f3 Bc8 34. Kf2 Bf5 35. Ra7 g6 36. Ra6+
Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6 Nf2
42. g4 Bd3 43. Re6
]]>
</sanimg>
<sanitimg>
<![CDATA[
1. e4 e5 2. Cf3 Cc6 3. Ab5 a6 4. Aa4 Cf6 5. O-O Ae7 6. Te1 b5
7. Ab3 d6 8. c3 O-O 9. h3 Cb8 10. d4 Cbd7 11. c4 c6 12. cxb5
axb5 13. Cc3 Ab7 14. Ag5 b4 15.  Cb1 h6 16. Ah4 c5 17. dxe5
Cxe4 18. Axe7 Dxe7 19. exd6 Df6 20. Cbd2 Cxd6 21.  Cc4 Cxc4
22. Axc4 Cb6 23. Ce5 Tae8 24. Axf7+ Txf7 25. Cxf7 Txe1+ 26. Dxe1
Rxf7 27. De3 Dg5 28. Dxg5 hxg5 29. b3 Re6 30. a3 Rd6 31. axb4
cxb4 32. Ta5 Cd5 33.  f3 Ac8 34. Rf2 Af5 35. Ta7 g6 36. Ta6+
Rc5 37. Re1 Cf4 38. g3 Cxh3 39. Rd2 Rb5 40. Td6 Rc5 41. Ta6 Cf2
42. g4 Ad3 43. Te6
]]>
</sanitimg>
</imgblock>
</object>

...
```

Figura 4.27. Codice incorporato SAN: nel primo diagramma i pezzi sono stati indicati secondo la denominazione in lingua inglese, nel secondo diagramma secondo la denominazione italiana.



Per esigenze più sofisticate di illustrazione scacchistica si può utilizzare l'elemento '**skakimg**':

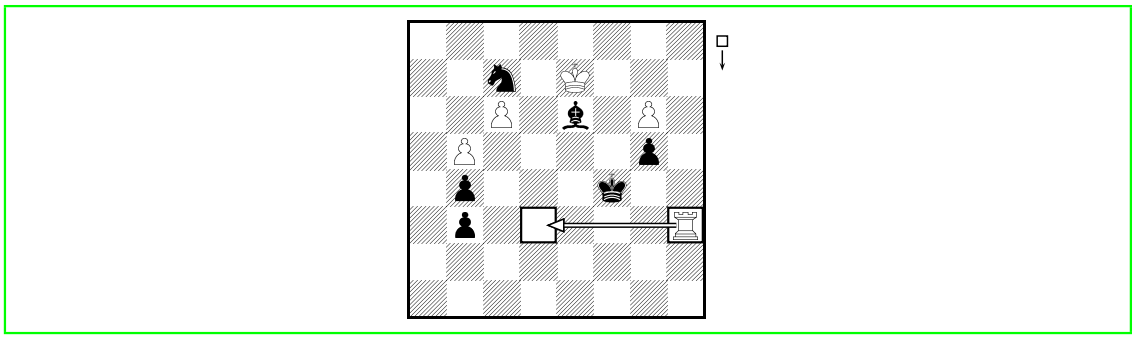
tale elemento consente di indicare situazioni scacchistiche di una certa complessità, comprese annotazioni grafiche sulla scacchiera, mediante i comandi definiti in TeX-Skak, un pacchetto per scrivere di scacchi in LaTeX.

Nel listato 4.28 viene presentato un esempio di incorporazione nell'ambito dell'elemento '**skakimg**'; corrispondentemente, la figura 4.29 illustra l'aspetto finale dell'immagine in relazione al codice presentato.

Listato 4.28. Codice incorporato TeX-Skak.

```
...

<object>
<imgblock>
<skakimg>
<![CDATA[
\fenboard{8/8/R5p1/2k2bp1/1p6/1P3PPn/3K4/8 b - - 0 41}
\hidemoves{41...Nf2 42. g4 Bd3}
\showmoverOn
\notationOff
\showinverseboard
\highlight{a6,e6}
\printarrow{a6}{e6}
]]>
</skakimg>
</imgblock>
</object>

...
```

Figura 4.29. Codice incorporato TeX-Skak.



4.5 Immagini incorporate Jgraph

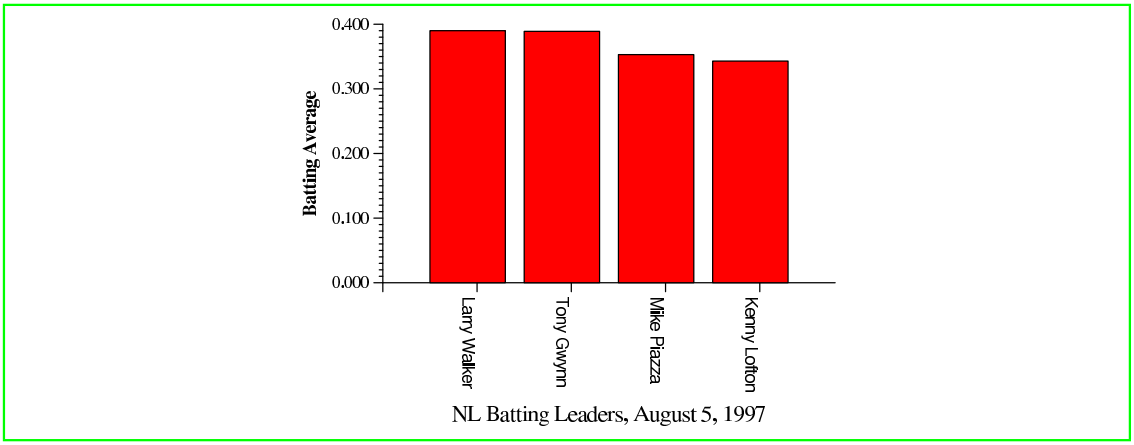
È possibile incorporare codice per tracciare grafici e diagrammi strutturati di vario genere attraverso l'elemento '**jgraphimg**', descrivendo l'immagine desiderata nel linguaggio di input del programma Jgraph.

Nel listato 4.30 viene presentato un esempio di incorporazione nell'ambito dell'elemento '**jgraphimg**'; corrispondentemente, la figura 4.31 illustra l'aspetto finale dell'immagine in relazione al codice presentato.

Listato 4.30. Codice incorporato Jgraph.

```
...

<object>
<imgblock>
<jgraphimg>
<![CDATA[
newgraph

xaxis
  min 0.1 max 4.9
  size 3.5
  hash 1 mhash 0 no_auto_hash_labels

yaxis
  min 0 max .4
  size 2
  precision 3

newcurve marktype xbar cfill 1 0 0 marksize .8
  pts   1 .390
        2 .389
        3 .353
        4 .343

xaxis
  hash_label at 1 : Larry Walker
  hash_label at 2 : Tony Gwynn
  hash_label at 3 : Mike Piazza
  hash_label at 4 : Kenny Lofton
  hash_labels hjl vjc font Helvetica rotate -90

yaxis label : Batting Average
title : NL Batting Leaders, August 5, 1997
]]>
</jgraphimg>
</imgblock>
</object>

...
```

Figura 4.31. Codice incorporato Jgraph.



4.6 Riferimenti

- *Algebraic chess notation*

  ⟨*http://en.wikipedia.org/wiki/Algebraic_chess_notation* ⟩

- *Forsyth-Edwards Notation*

  ⟨*http://en.wikipedia.org/wiki/Forsyth-Edwards_Notation* ⟩

[1] **GNU Plotutils** GNU GPL

Parte iii

Grafica per sopravvivere

Il mondo dell'informatica appare oramai indissolubilmente legato al settore della grafica e delle sue applicazioni, tanto più se si considera che (per la maggior parte degli utenti) l'uso di un sistema di elaborazione passa attraverso l'impiego di una shell grafica più o meno evoluta, e che il settore dell'intattenimento multimediale è sempre più dipendente dai sistemi automatici per le elaborazioni audio e video.

In questa parte si intendono suggerire e proporre punti di vista in proposito che potrebbero anche essere definiti «eretici». In un ottica che vuole considerare l'informatica non (o non solo) come uno dei più attivi ambiti del consumismo massificato, la grafica può rivelare anche dei lati forse inaspettati: concettuali, filosofici, addirittura educativi e pedagogici.

In particolare, verrà prestata particolare attenzione ai cosiddetti «minilinguaggi» grafici: si tratta di linguaggi (a volte tutt'altro che piccoli, ma tant'è...) orientati alla descrizione testuale del disegno che si desidera ottenere. Alcuni di questi linguaggi risalgono ai periodi «eroici» dell'informatica; dedicare questo spazio ad essi significa effettuare un'operazione di archeologia, recupero e conservazione culturale nel medesimo spirito che può avere ispirato Daniele Giacomini nelle parti dei suoi *Appunti di informatica libera* dedicate all'hardware e al software «d'annata». Un'ulteriore motivazione (di carattere più pratico che filosofico) è stata l'assoluta carenza di materiale in lingua italiana su alcuni di questi argomenti.

Nel seguito si tenterà di illustrare - o anche solo sfiorare - tali aspetti, con una particolare attenzione ad alcune possibili applicazioni didattiche della *computer graphics*.

Capitolo 5

Disegnare con Pic

La maggior parte delle persone è incapace di disegnare a mano libera anche una semplice linea retta, per non parlare di figure o grafici più complessi; tali persone, pertanto, tendono ad affidarsi a descrizioni verbali delle immagini che vorrebbero disegnare.[1] Con l'ausilio di qualcuno che sia dotato di capacità grafiche è allora possibile ottenere un disegno di buona qualità, a patto di riuscire a comunicare al disegnatore ciò che si intende con un sufficiente grado di chiarezza.

Pic è un preprocessore per Troff (v. sezione 5.23) mediante il quale è possibile ottenere immagini a partire da una descrizione testuale.

5.1 Riconoscimento storico

Il programma Pic originale fu scritto da Brian Kernighan, come complemento al programma Troff di Joseph Ossanna; fu successivamente riscritto da Kernighan, con miglioramenti sostanziali.

Il linguaggio trae ispirazione da linguaggi grafici più antichi, come Ideal e Grap. Kernighan dà credito a Chris van Wyk (l'autore di Ideal) per molte delle idee che sono poi confluite in Pic.

La descrizione originale del linguaggio Pic fu data da Kernighan in una relazione tecnica (vedi sezione 5.23) di cui esistono due revisioni (anno 1984 e anno 1991).

Esiste una realizzazione GNU di Pic, ossia Gpic, scritta da James Clark ⟨*jjc (ad) jclark·com* ⟩ come parte di Groff;[2] al momento della stesura del presente lavoro, la manutenzione è affidata a Ted Harding ⟨*ted·harding (ad) nessie·mcc·ac·uk* ⟩ e a Werner Lemberg ⟨*wl (ad) gnu·org* ⟩.

5.2 Natura di Pic

Pic è un preprocessore che estende le funzionalità di Troff al fine di consentire in modo semplice di tracciare il tipo di diagramma «a scatole e frecce» che si incontra comunemente negli articoli e nei libri di testo di argomento tecnico.

Il linguaggio Pic permette di descrivere in modo procedurale i diagrammi da tracciare, al fine di includere le immagini ottenute nei documenti generati da Troff (oppure da TeX o, in alternativa, per generare file grafici autonomi da utilizzare in seguito, come specificato nella sezione 5.22). Il linguaggio è abbastanza flessibile da consentire il tracciamento di diagrammi di stato, reti di Petri, [3] diagrammi di flusso, semplici schemi circuitali, mappe concettuali, [4] organigrammi e in generale qualsiasi tipo di illustrazione che richieda l'utilizzo ripetitivo di semplici forme geometriche e curve. La descrizione dei diagrammi è procedurale e orientata agli oggetti, pertanto produce una notazione compatta e semplice da modificare.

Il preprocessore Pic ha un duplice scopo: da un lato permette la descrizione e la generazione di semplici diagrammi e grafici mediante l'uso di un linguaggio naturale; dall'altro consente l'utlizzo di costrutti simili a un normale linguaggio di programmazione, consentendo di ottenere descrizioni molto sintetiche (naturalmente pagando un certo prezzo alla leggibilità; questa possibilità risulta in realtà particolarmente utile se si considera che è possibile preparare l'input per Pic utilizzando un programma esterno).

Il modo migliore per imparare Pic è quello di procedere per tentativi, osservare l'output, eventualmente apporre correzioni e modifiche al sorgente e iterare il procedimento sino al raggiungimento dell'obiettivo prefissato.

Essendo un preprocessore, Pic rielabora alcune specifiche parti di un sorgente Troff prima che quest'ultimo possa procedere ad ulteriori elaborazioni; in pratica Pic traduce queste parti in codici di formattazione di basso livello che Troff può a sua volta comprendere ed elaborare.

Esattamente come nel caso di Tbl e Eqn, una speciale coppia di macro delimita la parte di input che Pic deve elaborare, come segue:

```
.PS
descrizione_pic
.PE
```

Per l'utilizzo effettivo del preprocessore, si veda la sezione 5.22.

5.3 Versioni di Pic

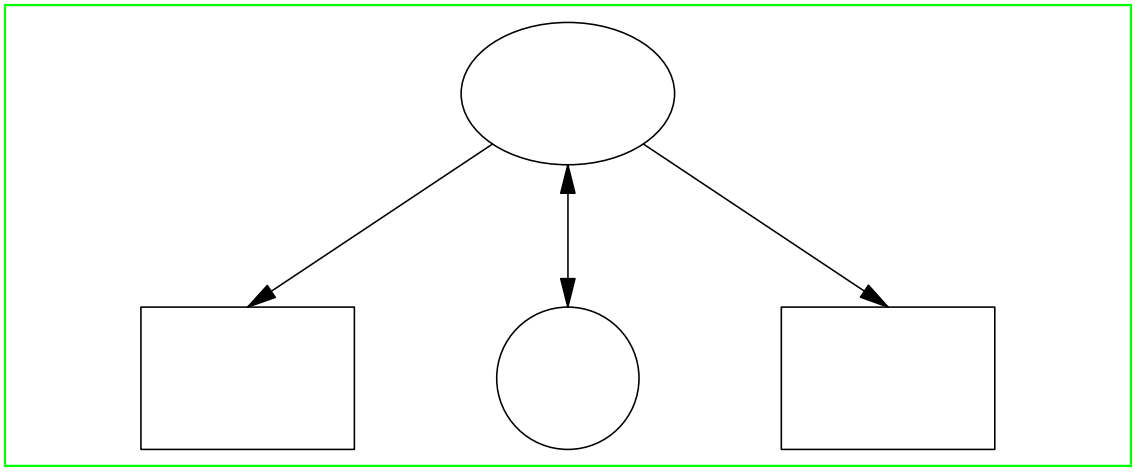
Pic nasce nell'anno 1984, ma la versione originale va oggi ritenuta obsoleta. Pic venne riscritto nell'anno 1991 come parte del pacchetto DBW (*Documenter's Workbench*) per lo Unix System V di AT&T.

Ove fosse necessario evidenziare le differenze fra Pic dell'anno 1991 e la realizzazione GNU (Gpic), verranno nel seguito utilizzati i nomi DWB Pic e GNU Gpic rispettivamente. In ogni caso gli esempi mostrati sono stati sperimentati tramite GNU Gpic.

5.4 Un esempio introduttivo

Si immagini di dover descrivere a un interlocutore telefonico la figura 5.1.

Figura 5.1. Primo esempio di figura Pic.



Si potrebbe dire:

> In alto c'è un'ellisse. Ci sono delle frecce collegate a due riquadri e anche a un cerchio...

Mettendosi nei panni di chi deve effettivamente tracciare il disegno dall'altro capo della linea telefonica, ci si rende subito conto delle difficoltà che sorgono se ci si sforza di essere precisi nella descrizione:

> Per prima cosa disegna un'ellisse. Poi scendi sotto l'ellisse e disegna un cerchio. Poi disegna un riquadro a sinistra, e un'altro identico a destra del cerchio. Poi traccia una freccia dal punto in basso a sinistra dell'ellisse al punto in alto del riquadro sinistro. Poi traccia una freccia dal punto in basso a destra dell'ellisse al punto in alto del riquadro destro. Infine traccia una doppia freccia che collega il punto in basso dell'ellisse con il punto in alto del cerchio.

Il listato 5.2 presenta la descrizione equivalente nel linguaggio Pic.

Listato 5.2. Primo esempio di descrizione Pic.

```
.PS
ellipse
move down from bottom of last ellipse
circle
move left from left of last circle
box
move right from right of last circle
box
arrow from lower left of last ellipse to top of 1st box
arrow from lower right of last ellipse to top of 2nd box
arrow <-> from bottom of last ellipse to top of last circle
.PE
```

Anche senza conoscere il linguaggio Pic, se si ha una certa familiarità con la lingua inglese è comunque possibile comprendere la descrizione. Si menzionano alcuni oggetti: un'ellisse ('**ellipse**'), due riquadri ('**box**'), un cerchio ('**circle**') e tre frecce ('**arrow**'). Vengono specificati degli spostamenti ('**move**') e dei cambiamenti di direzione ('**down**', '**left**', '**right**'). Inoltre vengono disposti alcuni oggetti relativamente alla posizione di alcuni altri, posizionando i riquadri a sinistra ('**from left of last circle**') e a destra ('**from right of last circle**') del cerchio e tracciando le frecce fra i vari oggetti ('**from lower left of last ellipse to top of 1st box**', '**from lower right of last ellipse to top of 2nd box**', '**<-> from bottom of last ellipse to top of last circle**').

Dall'esempio illustrato dovrebbe emergere il tono generale del linguaggio Pic. In generale si può affermare che la complessità della descrizione va di pari passo con la complessità della figura che si intende ottenere.

5.5 Concetti fondamentali

Un sorgente Pic è come un piccolo programma che contiene la ***descrizione*** di una figura. Pic compila tale programma traformandolo in delle macro per Troff. Di conseguenza, qualsiasi altro programma che debba elaborare l'output del programma Troff può tranquillamente ignorare il fatto che originariamente il sorgente conteneva codice Pic.

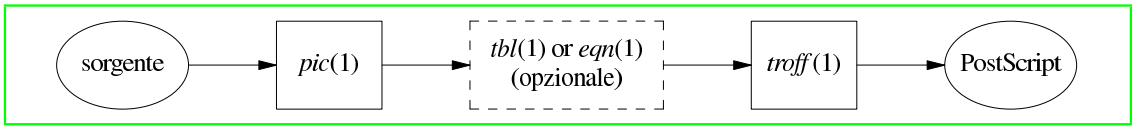
Pic tenta dunque di tradurre tutto ciò che si trova compreso fra le macro '**.PS**' e '**.PE**', lasciando invariato tutto il resto. In realtà, le suddette macro sono definite nel pacchetto di macro '**ms**' (e altrove) e hanno l'effetto di centrare orizzontalmente l'output di Pic.

Le immagini sono descritte in modo procedurale, come collezioni di oggetti collegati da movimenti. Normalmente, Pic cerca di allineare gli oggetti da sinistra a destra nell'ordine di descrizione, collegandoli in punti che siano visivamente naturali. Il listato 5.3 e la figura 5.4 illustrano, ad esempio, il flusso dei dati nell'elaborazione di un sorgente Pic.

Listato 5.3.

```
.PS
ellipse "sorgente";
arrow;
box width 0.6 "\fIpic\/\fP(1)"
arrow;
box width 1.1 "\fItbl\/\fP(1) or \fIeqn\/\fP(1)" "(opzionale)" dashed;
arrow;
box width 0.6 "\fItroff\/\fP(1)";
arrow;
ellipse "PostScript"
.PE
```

Figura 5.4. Il flusso dei dati nell'elaborazione di un sorgente (Pic).

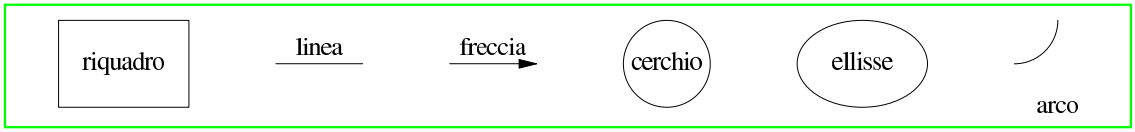


Il listato 5.3 illustra già parecchi concetti fondamentali di Pic: per prima cosa l'invocazione di tre degli oggetti fondamentali (ellisse, freccia, riquadro); poi la dichiarazione di righe di testo da inserire negli oggetti (si noti che si può anche indicare il tipo di carattere); poi l'indicazione di uno stile tratteggiato per un oggetto; infine l'allargamento di un riquadro per adattarlo al testo incluso.

Dal punto di vista sintattico, si noti che le istruzioni terminano con il carattere *newline* oppure con il carattere punto e virgola ('`;`'); gli argomenti testuali devono essere inclusi fra apici doppi ('`"`'); in generale l'ordine degli argomenti e dei modificatori (come '`width 1.2`' o '`dashed`') è ininfluente, tranne nel caso degli argomenti testuali.

La figura 5.5 presenta sei dei sette oggetti Pic fondamentali, alla loro dimensione predefinita.

Figura 5.5. Sei dei sette oggetti Pic fondamentali.



L'oggetto mancante è ***spline***; inoltre esiste una maniera per raccogliere vari oggetti in un ***blocco composto*** in modo da trattare per molti versi il gruppo come un oggetto singolo (similmente a un riquadro). Se ne parlerà nelle sezioni 5.7.3 e 5.13.2, rispettivamente.

Gli oggetti ***riquadro***, ***ellisse***, ***cerchio*** e ***blocco composto*** sono ***oggetti chiusi***; gli oggetti ***linea***, ***freccia***, ***arco*** e ***spline*** sono ***oggetti aperti***. Tale distinzione risulta importante per comprendere il funzionamento dei modificatori dei comandi.

La figura 5.5 è stata creata mediante il codice presentato nel listato 5.6, nel quale vengono introdotti alcuni altri concetti fondamentali.

Listato 5.6.

```
1   .PS
2   box "riquadro";
3   move;
4   line "linea" "";
5   move;
6   arrow "freccia" "";
7   move;
8   circle "cerchio";
9   move;
10  ellipse "ellisse";
11  move;
12  arc; down; move; "arco"
13  .PE
```

La prima cosa da notare è il comando '**move**', il quale esegue uno spostamento nella direzione corrente di una lunghezza predefinita (0,5 in).

Si noti inoltre come sia possibile arricchire linee e frecce mediante del testo. In questo esempio i comandi '**line**' e '**arrow**' ricevono entrambi due argomenti, i quali specificano rispettivamente il testo che andrà al di sopra e al di sotto dell'oggetto. Il motivo per cui un argomento testuale solo non va bene dovrebbe essere chiaro osservando l'output di '**arrow "Ahi!"**' (figura 5.7).

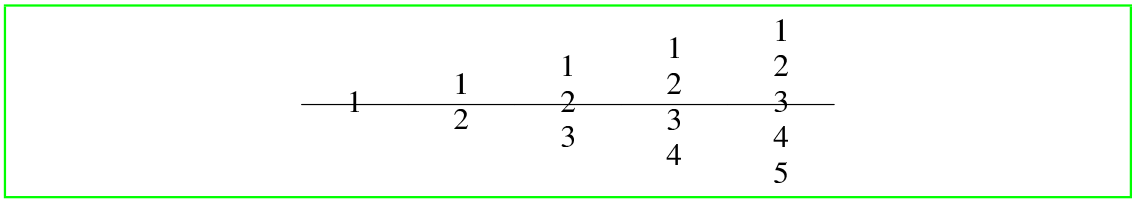
Figura 5.7. Testo centrato su una freccia.

Quando un comando riceve una stringa di testo, Pic tenta di posizionarla al centro geometrico dell'oggetto; aggiungendo più stringhe, si ottiene un blocco verticale che viene complessivamente centrato; si osservi ad esempio il codice del listato 5.8 e la corrispondente figura 5.9.

Listato 5.8.

```
.PS
line "1";
line "1" "2";
line "1" "2" "3";
line "1" "2" "3" "4";
line "1" "2" "3" "4" "5";
.PE
```

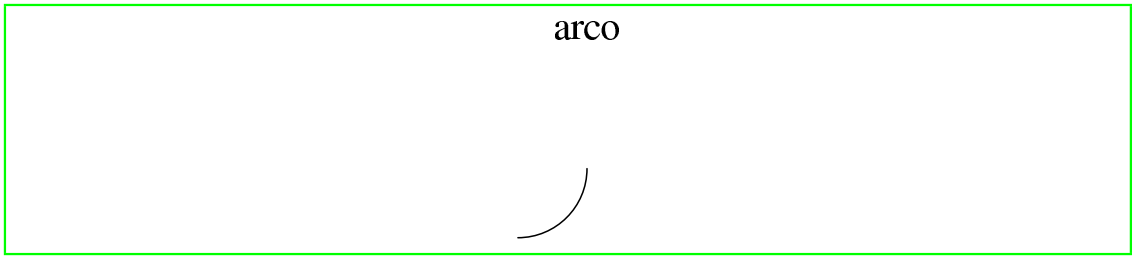
Figura 5.9. Effetto di argomenti testuali multipli.



La riga 12 del listato 5.6, la quale descrive l'arco con didascalia, introduce alcuni ulteriori concetti. Si osservi intanto come sia possibile variare la direzione di congiungimento fra i vari oggetti: omettendo l'argomento '**down**', la didascalia si sarebbe congiunta al di sopra dell'arco (figura

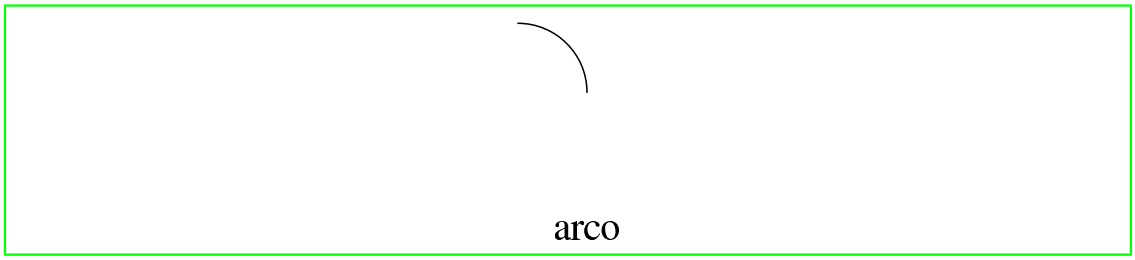
5.10).

Figura 5.10. Output di `**arc; move; "arco"**'.



La causa del suddetto comportamento risiede nel fatto che il tracciamento di un arco modifica la direzione corrente in quella verso cui punta il suo estremo finale. Se la cosa non fosse ancora chiara si consideri la figura 5.11.

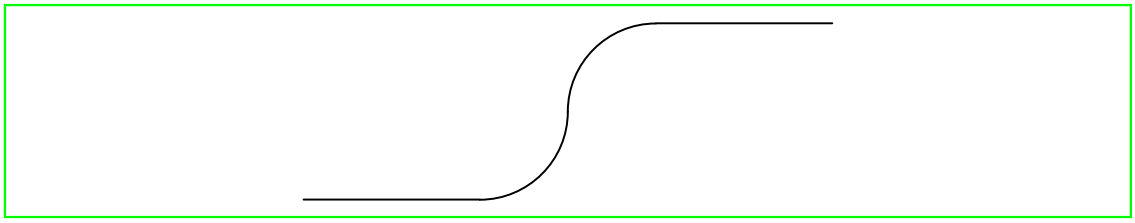
Figura 5.11. Output di `**arc cw; move; "arco"**'.



Nella figura 5.11 l'unica differenza rispetto alla figura 5.10 sta nell'uso dell'argomento '**cw**', che sta per *clockwise* ossia «verso orario» ('**ccw**' invece sta per *counter-clockwise* ossia «verso antiorario»). Si osservi come ciò cambi la direzione corrente verso il basso, piuttosto che verso l'alto.

Un'altra maniera per rendersi conto di quanto sopra osservato è presentata in figura 5.12.

Figura 5.12. Output di `**line; arc; arc cw; line**'.



Si noti che nella figura 5.12 non è stato necessario specificare la direzione verso l'alto per il secondo arco per ottenere una giunzione «liscia» con il primo.

Si osservi infine che una stringa isolata viene considerata come se fosse circondata da un riquadro invisibile dalle dimensioni specificate dagli attributi '**width**' e '**height**' oppure dai valori predefiniti delle variabili '**textwid**' e '**textht**' (il cui valore iniziale è zero per entrambe, visto che non si può prevedere la dimensione dei caratteri).

5.6 Dimensioni e spaziature

Le dimensioni sono implicitamente specificate in pollici (*inch*). Non volendo utilizzare i pollici, è possibile impostare la variabile '**scale**' per cambiare l'unità di misura. Impostando '**scale=2.54**' si passa in pratica da pollici a centimetri.[5]

5.6.1 Dimensioni predefinite

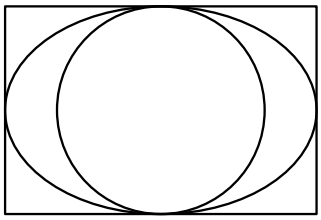
La tabella 5.13 presenta le dimensioni predefinite per gli oggetti Pic.

Tabella 5.13. Dimensioni predefinite per gli oggetti Pic.

Oggetto	Dimensioni predefinite (in pollici)
Riquadro (‘**box**’)	Larghezza: 0,75, altezza: 0,5
Cerchio (‘**circle**’)	Diametro: 0,5
Ellisse (‘**ellipse**’)	Larghezza: 0,75, altezza: 0,5
Arco (‘**arc**’)	Raggio: 0,5
Linea (‘**line**’)	Lunghezza: 0,5
Freccia (‘**arrow**’)	Lunghezza: 0,5

Il modo più intuitivo per considerare i valori predefiniti è osservare che tutti gli oggetti, per difetto, occupano comodamente l’interno di un riquadro standard (figura 5.14).

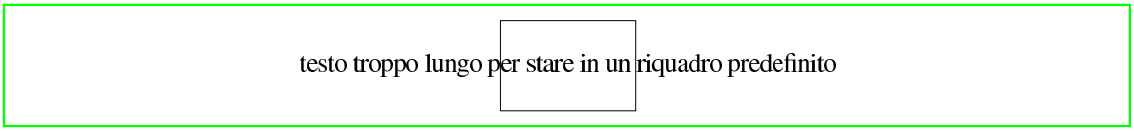
Figura 5.14. Output di ‘`box; circle at center of last box; ellipse at center of last box`’.



5.6.2 Gli oggetti non sono elastici

Il testo viene presentato nella fonte tipografica corrente con spaziatura predefinita da Troff. Riquadri, cerchi ed ellissi **non** si adattano automaticamente al testo racchiuso. Si osservi l’output di ‘`box "testo troppo lungo per stare in un riquadro predefinito"`’, presentato in figura 5.15: probabilmente non è ciò che di solito si intende ottenere.

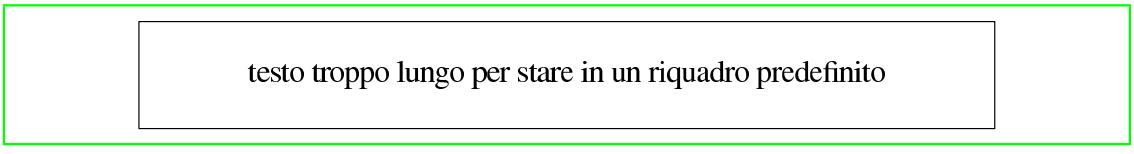
Figura 5.15. I riquadri non si adattano automaticamente.



5.6.3 Ridimensionamento dei riquadri

Per cambiare le dimensioni di un riquadro è possibile specificarne la larghezza mediante il modificatore ‘**width**’ (figura 5.16).

Figura 5.16. Output di ‘`box width 4 "testo troppo lungo per stare in un riquadro predefinito"`’.

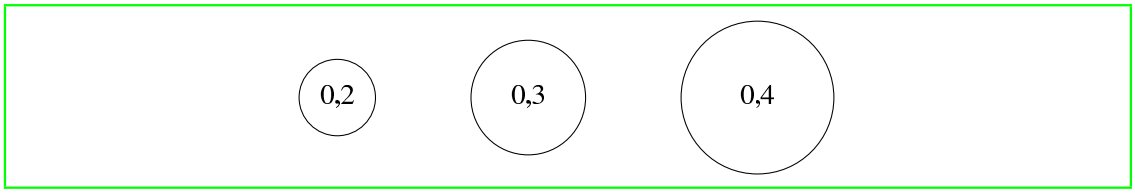


Il modificatore accetta una dimensione espressa in pollici. È previsto anche il modificatore ‘**height**’ il quale modifica l’altezza del riquadro. Le parole chiave ‘**width**’ e ‘**height**’ possono essere abbreviate mediante ‘**wid**’ e ‘**ht**’ rispettivamente.

5.6.4 Ridimensionamento di altri oggetti

Per modificare le dimensioni di un cerchio si utilizzano i modificatori '**radius**' o '**diameter**' (abbreviabili in '**rad**' o '**diam**', rispettivamente); a seconda del valore numerico successivamente indicato si ottiene una modifica del raggio o del diametro del cerchio, rispettivamente (figura 5.17).

Figura 5.17. Cerchi di raggio crescente.

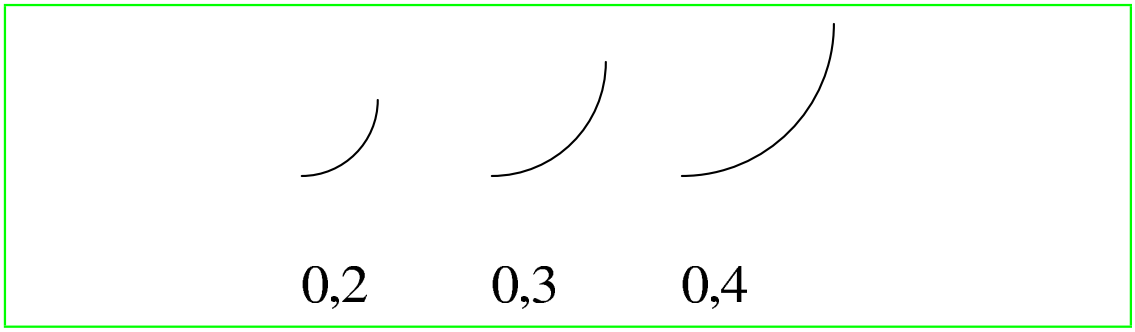


Anche il comando '**move**' accetta una dimensione, la quale indica di quanti pollici ci si deve spostare nella direzione corrente.

Le ellissi hanno dimensioni tali da essere inscritte nel riquadro rettangolare definito dai loro assi, e possono essere ridimensionate tramite '**width**' e '**height**', come i riquadri.

È possibile modificare il raggio di curvatura di un arco tramite '**radius**' (o '**rad**'), il quale specifica il raggio del cerchio cui l'arco appartiene. Al crescere del valore indicato si ottengono archi più piatti (figura 5.18).

Figura 5.18. Archi di raggio crescente.

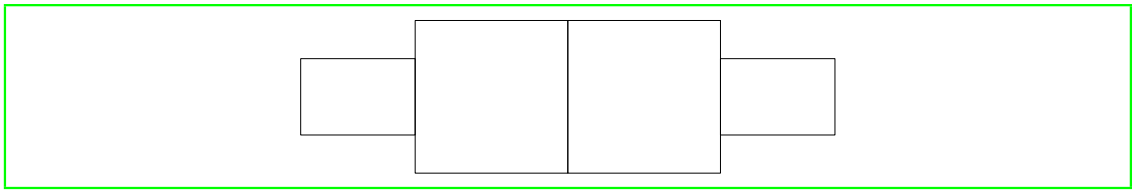


Si noti che, a causa del fatto che un arco equivale a un quarto di cerchio, aumentando il raggio si aumenta anche la dimensione del riquadro circoscritto all'arco.

5.6.5 La parola chiave «same»

Al posto dell'indicazione esplicita delle dimensioni è possibile utilizzare la parola chiave '**same**'. In tal modo l'oggetto avrà le medesime dimensioni dell'oggetto (dello stesso tipo) che precede (figura 5.19).

Figura 5.19. Output di '`box; box wid 1 ht 1; box same; box`'.

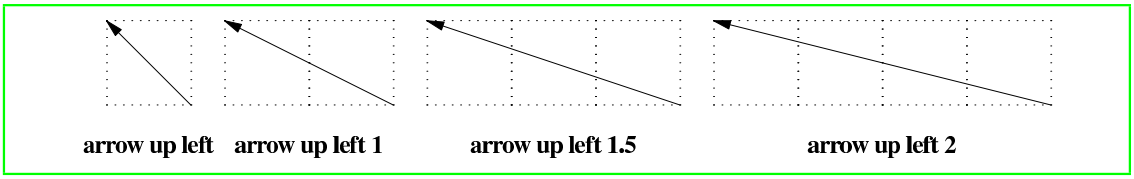


5.7 Linee generalizzate e spline

5.7.1 Linee diagonali

È possibile specificare linee o frecce diagonali aggiungendo uno o più modificatori '**`up`**', '**`down`**', '**`left`**' oppure '**`right`**' all'oggetto. Ognuno dei suddetti modificatori può presentare un coefficiente moltiplicatore. Per comprenderne l'effetto si immagini la superficie grafica con una griglia di riquadri standard (figura 5.20).

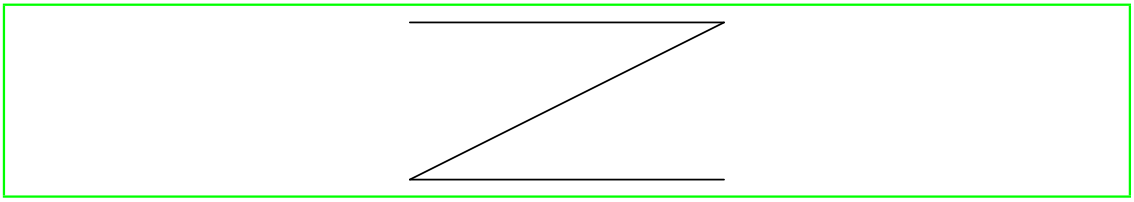
Figura 5.20. Frecce diagonali (i riquadri punteggiati mostrano la griglia sottintesa, spaziata di mezzo pollice).



5.7.2 Spezzate

Un oggetto linea o freccia può anche consistere di una spezzata composta da segmenti di diversa lunghezza e direzione. Per descrivere una spezzata si possono collegare diversi comandi '**`line`**' o '**`arrow`**' mediante la parola chiave '**`then`**' (figura 5.21).

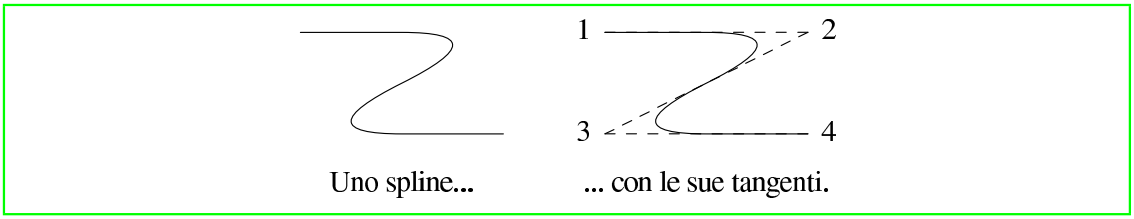
Figura 5.21. Output di '`line right 1 then down .5 left 1 then right 1`'.



5.7.3 Spline

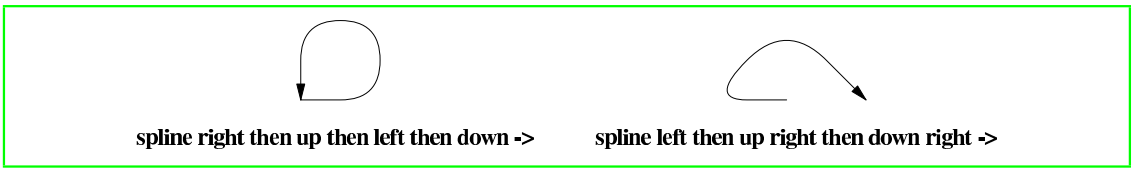
Cominciando una spezzata con la parola chiave '**`spline`**' si ottiene che i vertici della spezzata vengano considerati punti di controllo per una curva regolare interpolatrice (figura 5.22).

Figura 5.22. '`spline right 1 then down .5 left 1 then right 1`'.



Mediante gli spline è possibile descrivere molte curve irregolari ma dall'aspetto naturale (figura 5.23).

Figura 5.23. Altri due spline di esempio.



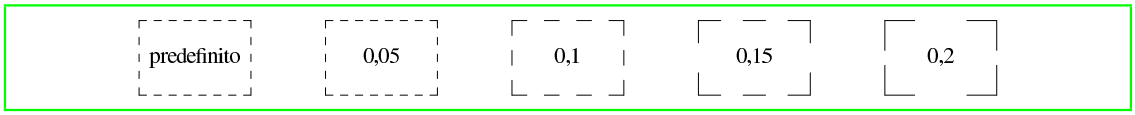
Si noti nella figura 5.23 la decorazione a freccia (‘**->**’). La punta della freccia si può applicare in modo naturale agli oggetti a spezzata, sia basati su linee che su spline (si veda in proposito la sezione 5.8).

5.8 Decorazioni

5.8.1 Oggetti tratteggiati

È stato già osservato che il modificatore ‘**dashed**’ può cambiare lo stile delle linee di un oggetto da solido a tratteggiato. GNU Gpic permette di punteggiare o tratteggiare ellissi,[6] cerchi e archi (e anche gli spline in modalità TeX, vedi sezione 5.20); alcune versioni di DWB Pic permettono solamente il tratteggio di linee e riquadri. È anche possibilile variare l’intervallo del tratteggio specificando un valore numerico subito dopo il modificatore (figura 5.24).

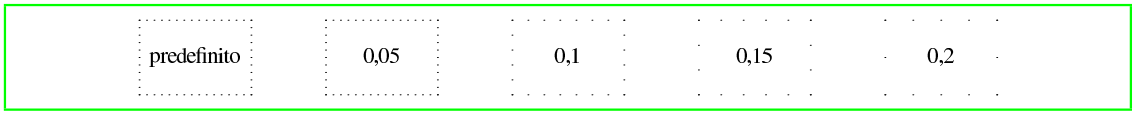
Figura 5.24. Oggetti tratteggiati.



5.8.2 Oggetti punteggiati

Un ulteriore modificatore disponibile è ‘**dotted**’. Anche in questo caso è possibile specificare l’intervallo di punteggiatura mediante un valore numerico subito dopo il modificatore (figura 5.25).

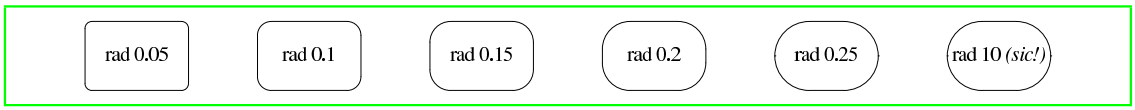
Figura 5.25. Oggetti punteggiati.



5.8.3 Vertici arrotondati

GNU Gpic permette anche l’arrotondamento degli vertici di un riquadro, mediante il modificatore ‘**rad**’ (figura 5.26).

Figura 5.26. ‘**box rad**’ con valori crescenti del valore numerico.

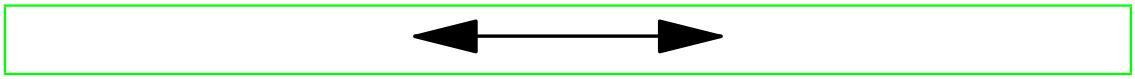


Valori del raggio di curvatura maggiori della metà della più piccola dimensione del riquadro vengono implicitamente troncati a tale valore.

5.8.4 Punte di freccia

Linee ed archi possono essere a loro volta decorati. Qualsiasi linea o arco (o spline) può essere decorata mediante punte di freccia utilizzando i modificatori '**->**' e '**<-**' (figura 5.27).

Figura 5.27. Doppia freccia realizzata mediante '**line <- ->**'.



Effettivamente il comando '**arrow**' altro non è che un sinonimo di '**line ->**'. Esiste inoltre il modificatore a doppia punta di freccia '**<->**', pertanto la figura 5.27 si può generare col codice '**line <->**'.

Le punte di freccia hanno un attributo '**width**', che specifica la dimensione trasversale della punta, e un attributo '**height**' che ne specifica la dimensione longitudinale.

Lo stile delle punte di freccia è controllato dalla variabile di stile (vedi sezione 5.14) denominata '**arrowhead**'. Le versioni DWB Pic e GNU Gpic interpretano tale variabile in maniera differente. Per difetto, DWB Pic usa punte di freccia aperte, con valore 2 per la variabile '**arrowhead**'; l'articolo di Kernighan (v. 5.23) specifica che un valore pari a 7 corrisponde a punte di freccia solide. Viceversa, per difetto, GNU Gpic usa punte di freccia solide e il valore 1 per '**arrowhead**'; il valore 0 corrisponde a punte di freccia aperte.[7]

5.8.5 Spessori

È anche possibile modificare lo spessore delle linee che compongono un oggetto (si tratta di un'estensione GNU Gpic; DWB Pic **non** prevede tale possibilità). Lo spessore predefinito è controllato dalla variabile '**linethick**'. Lo spessore è espresso in punti. Valori negativi implicano spessore predefinito: in modalità TeX (v.sezione 5.20), usando l'opzione '**-t**', ciò significa utilizzare uno spessore pari a otto millesimi di pollice; in modalità TeX, usando l'opzione '**-c**', significa lo spessore specificato dal comando '**.ps**' (*Point size*); in modalità Troff significa utilizzare uno spessore proporzionale alle dimensioni dei punti. Valore zero significa tracciare la linea con lo spessore minimo compatibile con il dispositivo di output. Il valore iniziale di '**linethick**' è -1. Esiste anche l'attributo '**thickness**' (abbreviabile in '**thick**'). Per esempio '**circle thickness 1.5**' traccia un cerchio con uno spessore di un punto e mezzo. Lo spessore delle linee non è influenzato dal valore della variabile '**scale**' (sezione 5.6), né dagli argomenti specificati assieme alla macro '**.PS**' (sezione 5.19).

5.8.6 Oggetti invisibili

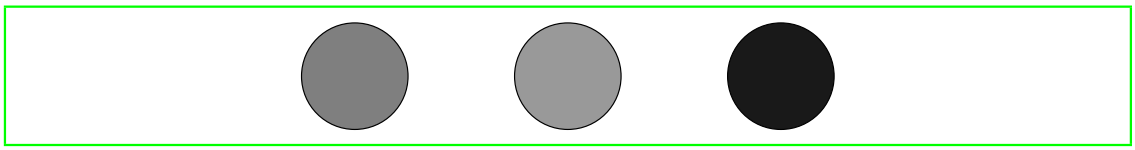
Il modificatore '**invisible**' (abbreviabile in '**invis**') rende un oggetto completamente invisibile. Un uso tradizionale di questa possibilità era quello di piazzare del testo in un oggetto invisibile, per ottenere un posizionamento naturale rispetto agli oggetti circostanti. Le versioni più recenti di DWB Pic e la realizzazione GNU Gpic gestiscono il testo isolato esattamente in questo modo.

5.8.7 Oggetti campiti

È possibile campire riquadri, cerchi ed ellissi mediante il modificatore '**`filled`**' (abbreviabile in '**`fill`**'). Si può anche specificare un valore numerico; il valore predefinito è dato dalla variabile di stile (vedi sezione 5.14) denominata '**`fillval`**'.

DWB Pic e GNU Gpic seguono convenzioni opposte per i valori di campitura, e prevedono valori predefiniti differenti. Per DWB Pic il valore predefinito di '**`fillval`**' è 0,3 e i valori più piccoli corrispondono a campiture più scure; GNU Gpic utilizza zero per il bianco e uno per il nero, e il valore predefinito di '**`fillval`**' è 0,5 (figura 5.28).

Figura 5.28. '`circle fill; move; circle fill 0.4; move; circle fill 0.9;`'.



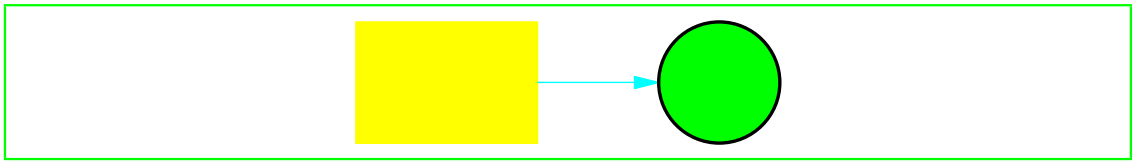
GNU Gpic fornisce ulteriori garanzie. È possibile usare un valore di campitura superiore a uno: ciò significa campire con la sfumatura di grigio attualmente usata per il testo e le linee. Ciò corrisponde normalmente al nero, ma certi dispositivi di output permettono di modificare tale comportamento. L'attributo di invisibilità non influisce la campitura degli oggetti. Qualsiasi testo associato con un oggetto campito viene disegnato dopo che l'oggetto è stato campito, sicché il testo non viene coperto dalla campitura.

Gli oggetti chiusi prevedono il modificatore '**`solid`**', che equivale a campire con il valore di campitura più scuro

5.8.8 Oggetti colorati

Come estensione GNU sono previsti altri tre modificatori per specificare il colore degli oggetti. '**`outline`**' imposta il colore del contorno, '**`shaded`**' il colore di campitura, '**`color`**' entrambi i colori. Tutti e tre prevedono un suffisso che specifichi il colore (figura 5.29).

Figura 5.29. '`box color "yellow"; arrow color "cyan"; circle shaded "green" thickness 1 outline "black";`'.



È prevista anche una sintassi alternativa: '**`colour`**', '**`colored`**', '**`coloured`**', '**`outlined`**'.

Al momento di questa stesura i colori non sono contemplati in modalità TeX (sezione 5.20). I nomi standard dei colori per Groff si trovano nei file di macro per i vari dispositivi di output (per esempio '`/usr/share/groff/current/tmac/ps.tmac`' per il dispositivo PostScript).

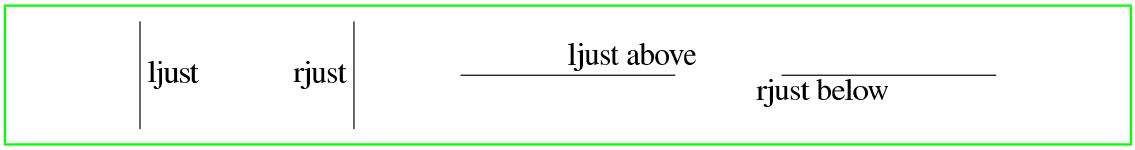
Pic presuppone che all'inizio di una figura sia il colore di tracciamento che quello di campitura siano impostati al valore predefinito.

5.9 Ulteriori indicazioni sul posizionamento del testo

Per difetto il testo viene centrato in corrispondenza del centro geometrico dell'oggetto a cui il testo è associato. Il modificatore '**`ljust`**' fa sì che l'estremo sinistro del testo si trovi nel punto specificato (il che significa che il testo giacerà **alla destra** del punto stesso), mentre il modificatore '**`rjust`**' fa sì che l'estremo destro del testo si trovi nel punto specificato. I modificatori '**`above`**' e '**`below`**' centrano il testo al di sopra e al di sotto dell'oggetto, rispettivamente, a una distanza pari a mezza linea di testo.

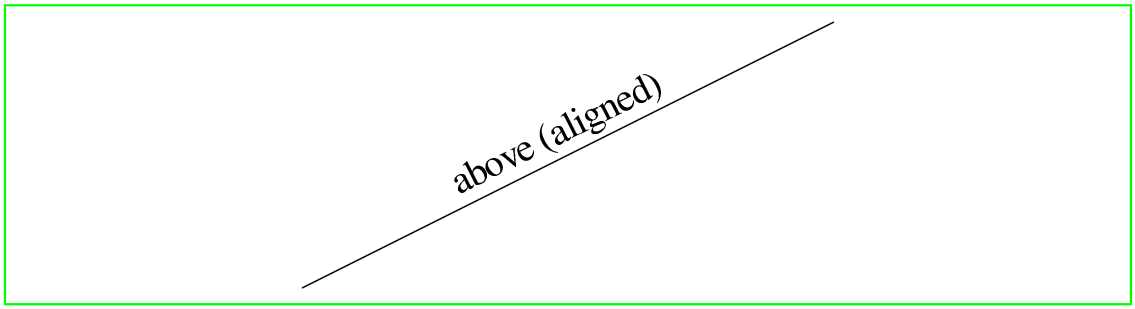
Gli attributi suddetti possono anche essere combinati (figura 5.30).

Figura 5.30. Attributi del testo.



In GNU Gpic gli oggetti possono avere un attributo '**`aligned`**', che ha effetto solo quando il postprocessore è '**`grops`**'.[8] Il testo associato a un oggetto con attributo '**`aligned`**' viene ruotato (figura 5.31) attorno al centro dell'oggetto in modo da risultare allineato nella direzione che va dal punto iniziale al punto finale dell'oggetto stesso.[9]

Figura 5.31. '**`line aligned up 1 right 2 "above (aligned)" above`**'.



5.10 Ulteriori indicazioni sui cambiamenti di direzione

Si è già spiegato come cambiare la direzione, lungo la quale gli oggetti vengono composti, da verso destra a verso il basso. Le figure 5.32 e 5.33 illustrano ulteriormente la cosa.

Figura 5.32. Effetti delle diverse direzioni di spostamento (verso destra e verso sinistra).

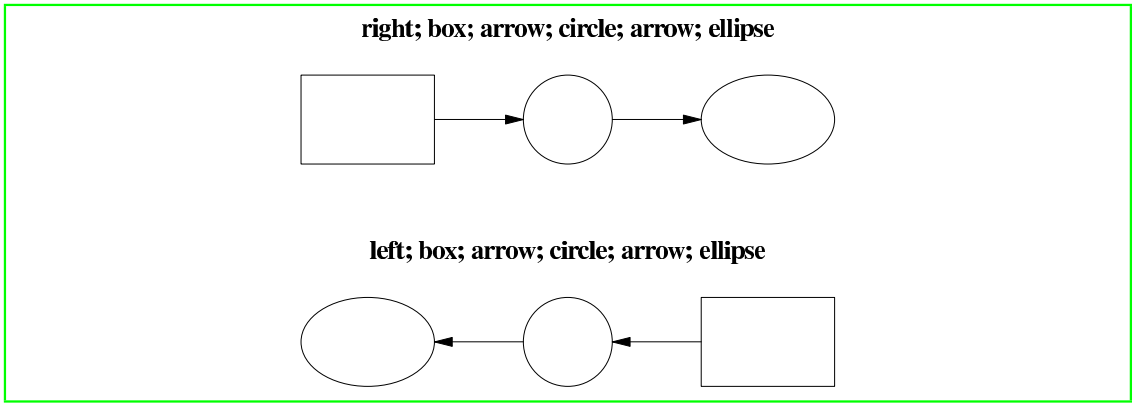
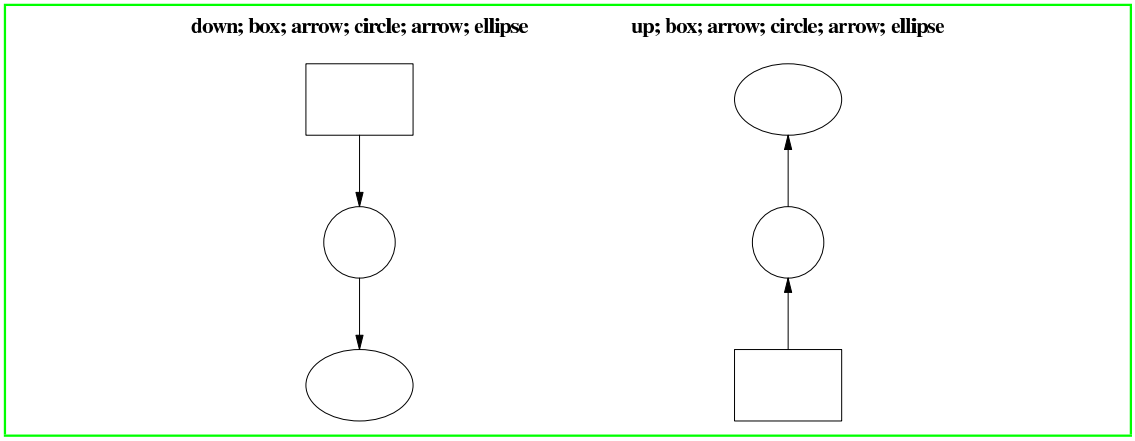
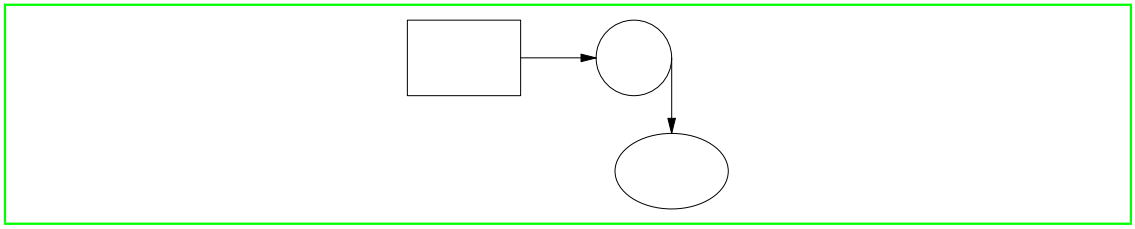


Figura 5.33. Effetti delle diverse direzioni di spostamento (verso il basso e verso l'alto).



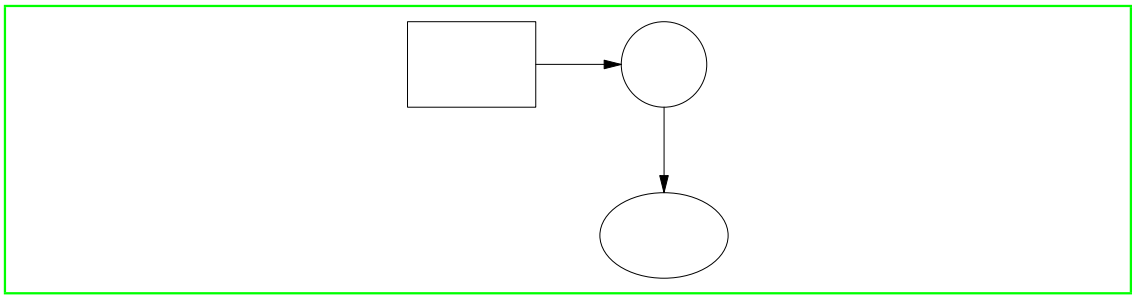
Può accadere qualcosa di sorprendente se si cambia direzione in modo apparentemente ovvio (figura 5.34).

Figura 5.34. '**box; arrow; circle; down; arrow; ellipse**'.



Magari ci si aspettava che il codice '**box; arrow; circle; down; arrow; ellipse**' producesse la figura 5.35, ma effettivamente per ottenere tale risultato si deve utilizzare il codice presentato nel listato 5.36. Per quale motivo? Perché il punto di uscita per la direzione corrente è già impostato quando si disegna il nuovo oggetto. La seconda freccia nella figura 5.34 cala verso il basso a partire dal punto del cerchio a cui si dovrebbe congiungere un nuovo oggetto verso destra.

Figura 5.35. Più intuitivo?



Listato 5.36.

```
1  .PS
2  box;
3  arrow;
4  circle;
5  move to bottom of last circle;
6  down;
7  arrow;
8  ellipse
9  .PE
```

Il significato di ‘**move to bottom of last circle;**’ dovrebbe essere evidente. Per comprendere la cosa in generale è necessario approfondire due temi importanti: posizioni e nomi degli oggetti (sezioni 5.12 e 5.11 rispettivamente).

5.11 Nomi

La maniera più naturale per indicare le posizioni in Pic è in relativamente agli oggetti. Per far ciò è necessario poter assegnare dei nomi agli oggetti stessi. Il linguaggio Pic prevede numerose possibilità in tal senso, mutuate in modo naturale della lingua inglese.

5.11.1 Nominare gli oggetti per ordine di tracciamento

Il modo più semplice (e in genere il più utile) per nominare un oggetto è tramite la clausola ‘**last**’. È necessario che essa sia seguita dall’indicazione del tipo di oggetto: ‘**box**’, ‘**circle**’, ‘**ellipse**’, ‘**line**’, ‘**arrow**’, ‘**spline**’, ‘**""**’ oppure ‘**[]**’ (l’ultimo tipo si riferisce a un ***blocco composto***, di cui si discuterà nella sezione 5.13.2). Pertanto, ad esempio, la clausola ‘**last circle**’ nel listato 5.36 si riferisce all’ultimo cerchio tracciato.

Generalizzando, si può dire che gli oggetti di un dato tipo sono implicitamente numerati a partire da 1. Ci si può riferire per esempio alla terza ellisse nella figura corrente tramite ‘**3rd ellipse**’, oppure al primo riquadro tramite ‘**1st box**’, o ancora alla quinta stringa di testo (la quale non sia attributo di un altro oggetto, naturalmente) tramite ‘**5th ""**’.

Inoltre gli oggetti sono numerati alla rovescia, per ciascun tipo, a partire dall’ultimo. Si può quindi utilizzare ‘**2nd last box**’ per indicare il penultimo riquadro, oppure ‘**3rd last ellipse**’ per la terz’ultima ellisse.

Ove sia prevista la notazione seguente:

```
nth
```

si può usare anche la seguente:

```
'espressione'th
```

ove ***espressione*** abbia valore numerico (intero).[10] Per un esempio, si consideri il listato 5.37.

Listato 5.37.

```
...
for i = 1 to 4 do {
   line from upper left of ‘i’th box to lower right of ‘i+1’th box
}
...
```

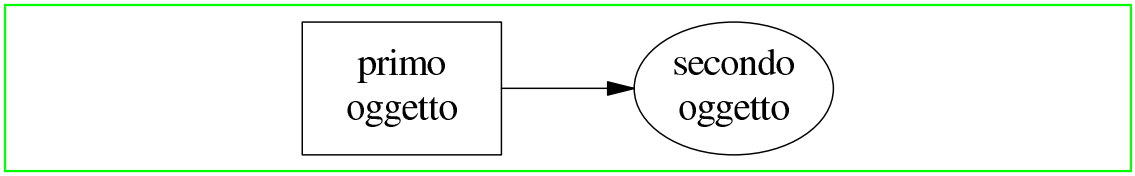
5.11.2 Nominare gli oggetti mediante etichette

Si possono menzionare gli oggetti anche tramite delle etichette. Un'***etichetta*** è una parola (che inizi con una lettera maiuscola) seguita dai due punti ('**:**'); viene dichiarata semplicemente posizionandola immediatamente prima del comando di tracciamento dell'oggetto. Ad esempio, il codice del listato 5.38 dichiara le etichette '**A**' e '**B**' per il primo e il secondo oggetto, rispettivamente; la figura 5.39 presenta il risultato.

Listato 5.38.

```
1  .PS
2  A: box "primo" "oggetto"
3  move;
4  B: ellipse "secondo" "oggetto"
5  move;
6  arrow right at right of A;
7  .PE
```

Figura 5.39. Esempio di utilizzo delle etichette.



L'istruzione '**at**' nella riga 6 utilizza l'etichetta '**A**' (il comportamento di '**at**' verrà spiegato nella sezione 5.12). In seguito (sezione 5.13.2) si evidenzierà quanto le etichette si rivelino particolarmente utili nel caso degli oggetti composti.

Le etichette non sono costanti, bensì variabili.[11] Con del codice come '**A: A + (1,0);**' si ottiene l'effetto di riassegnare all'etichetta '**A**' un valore che designa una nuova posizione che si trova un pollice a destra della vecchia (v. anche sezione 5.12).

5.12 Posizioni

La posizione dei punti sulla superficie grafica può venire descritta in molti modi differenti. Per quanto riguarda la sintassi del linguaggio Pic, le varie forme descrittive sono equivalenti: ove se ne possa usare una in particolare, ogni altra che abbia la stessa semantica è permessa anch'essa.

L'etichetta speciale '**Here**' si riferisce sempre alla posizione corrente.

5.12.1 Coordinate assolute

Il modo più immediato per indicare un punto è utilizzare le coordinate cartesiane assolute;[12] Pic utilizza un sistema di riferimento cartesiano con origine nel vertice inferiore sinistro della superficie grafica virtuale (con gli assi orientati in maniera usuale). Una posizione assoluta si può sempre scrivere nel modo usuale come coppia di numeri separati dalla virgola e racchiusi in parentesi (tonde) - e tale pratica è raccomandata per garantire la leggibilità del codice. Se il contesto è tale da escludere ambiguità, la coppia di coordinate può essere indicata senza le parentesi.

L'utilizzo delle coordinate assolute **è tuttavia sconsigliabile**, poiché esse tendono a rendere le descrizioni difficilmente comprensibili e adattabili. È meglio utilizzare la flessibilità del linguaggio Pic per specificare le posizioni **relativamente agli oggetti** precedentemente tracciati (sezione 5.12.2).

5.12.2 Posizioni relative agli oggetti

Il simbolo '**`Here`**' si riferisce sempre alla posizione dell'ultimo oggetto tracciato, oppure al punto finale dell'ultimo spostamento.

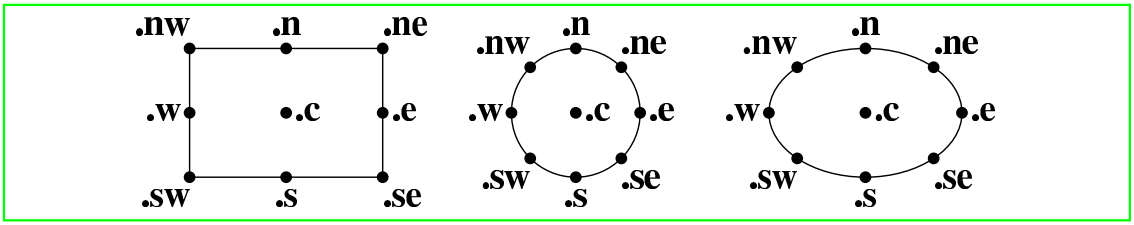
Senza ulteriori specificazioni, una descrizione come '**`last circle`**' come di qualunque oggetto chiuso o arco, si riferisce al centro geometrico dell'oggetto stesso. Senza ulteriori specificazioni, il nome di una linea o spline si riferisce al punto iniziale dell'oggetto stesso.

Inoltre gli oggetti Pic posseggono un certo numero di posizioni menzionabili associate ad essi. Una di queste è il centro dell'oggetto, il quale (per ridondanza) può essere indicato con il suffisso '**`.center`**' (o semplicemente '**`.c`**'). Sicché '**`last circle .center`**' equivale a '**`last circle`**'.

5.12.2.1 Posizioni relative agli oggetti chiusi

Ciascun oggetto chiuso (riquadri, cerchi, ellissi o blocchi composti) possiede altresì ben otto punti (cardinali) associato ad esso (figura 5.40).

Figura 5.40. I punti cardinali.



Pertanto, il codice '**`last circle .s`**' si riferisce al punto sud dell'ultimo cerchio tracciato. La riga 5 del listato 5.36 si potrebbe dunque scrivere così: '**`move to last circle .s;`**'.

In alternativa ai quattro punti cardinali principali ('**`.n`**', '**`.s`**', '**`.e`**' e '**`.w`**') sono disponibili rispettivamente i nomi '**`.top`**', '**`.bottom`**', '**`.left`**' e '**`.right`**' (o semplicemente '**`.t`**', '**`.b`**', '**`.l`**' e '**`.r`**').

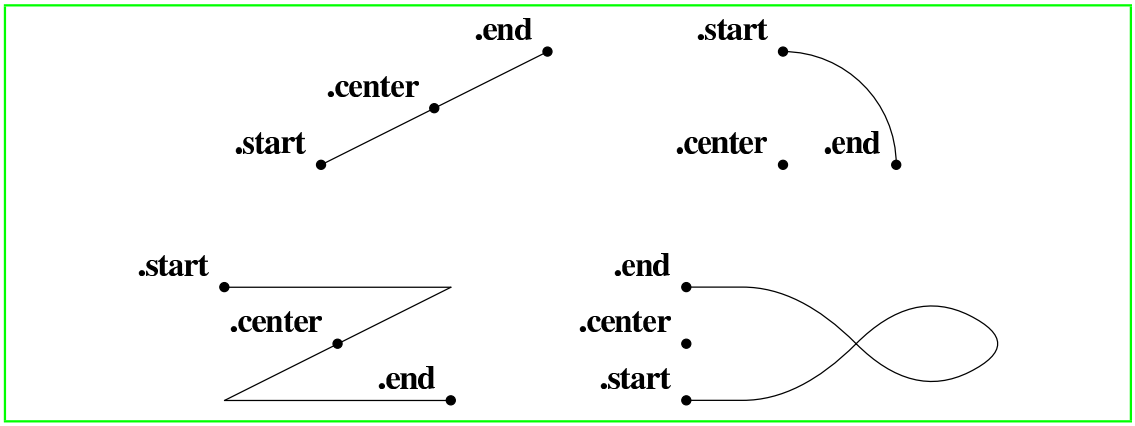
Si possono anche usare i nomi '**`center`**', '**`top`**', '**`bottom`**', '**`left`**', '**`right`**', '**`north`**', '**`south`**', '**`east`**' e '**`west`**' (si noti che manca il punto iniziale) in forma di prefisso marcato dalla parola chiave '**`of`**'; sicché '**`center of last circle`**' oppure '**`top of 2nd last ellipse`**' sono entrambi corretti riferimenti a oggetti. Infine, i nomi '**`left`**' e '**`right`**' possono essere preceduti da '**`upper`**' oppure '**`lower`**', con ovvio significato (vedi anche listato 5.2).

Anche gli archi possiedono punti cardinali, che concidono con quelli dell'associato cerchio.

5.12.2.2 Posizioni relative agli oggetti aperti

Gli oggetti aperti (linee, frecce, archi e spline) possiedono tre punti menzionabili: '**`.start`**', '**`.center`**' (oppure '**`.c`**') e '**`.end`**'. È possibile anche utilizzarli in forma prefissa (senza il punto iniziale, e con la parola chiave '**`of`**'). Il centro dell'arco coincide con il centro del cerchio associato, ma il centro di una linea, spezzata o spline è **il punto medio degli estremi** (figura 5.41).

Figura 5.41. Punti speciali associati agli oggetti aperti.



5.12.3 Comporre le posizioni

A partire da due posizioni date, esistono diverse maniere per combinarle insieme per contruirne una terza.

5.12.3.1 Somma vettoriale e traslazioni

Le posizioni possono essere addizionate o sottratte per generare una nuova posizione.[13] Il risultato è l'usuale somma vettoriale (somma per coordinate). Ad esempio, '**`last box .ne + (0.1, 0)`**' è una posizione valida: questo esempio costituisce un utilizzo tipico, per definire una posizione lievemente sfasata rispetto a quella data.[14]

5.12.3.2 Interpolazione

Una posizione può risultare dall'***interpolazione*** di due posizioni date. La sintassi è:

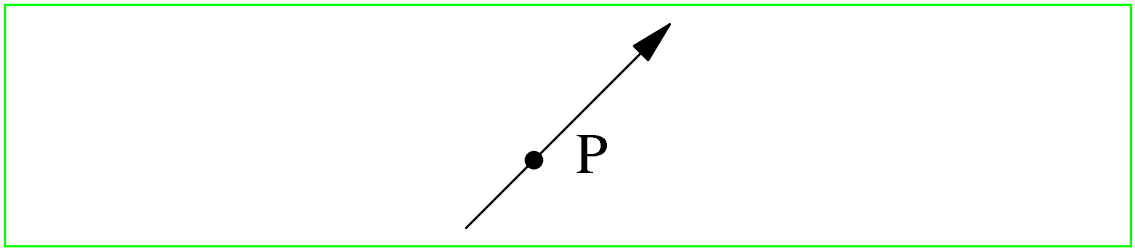
```
frazione of the way between prima_posizione and seconda_posizione
```

Per esempio, si può scrivere: '**`1/3 of the way between here and last ellipse .ne`**'. La frazione può esprimersi nella forma ***numeratore*/*denominatore*** oppure in notazione decimale.[15] Alternativamente, e in modo più sintetico, si può scrivere:

```
frazione <prima_posizione , seconda_posizione>
```

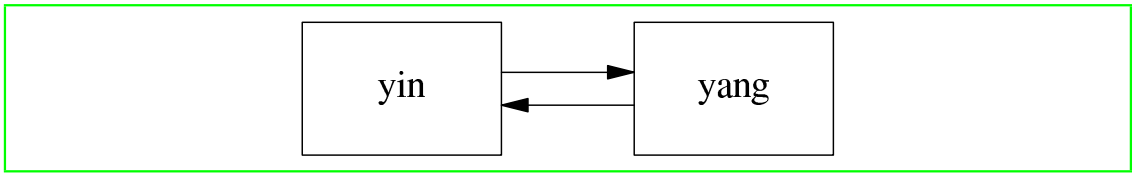
Pertanto il precedente esempio potrebbe scriversi: '**`1/3 <here, last ellipse .ne>`**'. Come ulteriore esempio si consideri la figura 5.42.

Figura 5.42. `P: 1/3 of the way between last arrow .start and last arrow .end'.



L'interpolazione può essere utile, ad esempio, per tracciare connessioni bidirezionali (figura 5.43).

Figura 5.43. Connessioni bidirezionali.



La figura 5.43 corrisponde al codice del listato 5.44. Si noti l'uso della forma abbreviata per l'interpolazione.

Listato 5.44.

```
A: box "yin"; move;
B: box "yang";
arrow right at 1/4 <A.e,A.ne>;
arrow left  at 1/4 <B.w,B.sw>;
```

5.12.3.3 Proiezioni

Date due posizioni ***prima_posizione*** e ***seconda_posizione***, la posizione:

(***prima_posizione*** , ***seconda_posizione***)

ha l'ascissa di ***prima_posizione*** e l'ordinata di ***seconda_posizione***. Ciò può tornare utile per posizionare oggetti ai vertici di un riquadro ipotetico determinato da altri due oggetti (figura 5.45).

Figura 5.45. Utilizzo della composizione per proiezione.

5.12.4 Utilizzo delle posizioni

Ci sono quattro modi di utilizzo delle posizioni: '**at**', '**from**', '**to**' e '**with**'. Si tratta di quattro modificatori, cioè vanno usati come suffisso dei comandi di tracciamento.

Il modificatore '**at**' comporta il tracciamento di un oggetto chiuso oppure di un arco con il centro coincidente con la posizione che segue, ossia il tracciamento di una linea, spline o freccia a partire dalla posizione che segue.

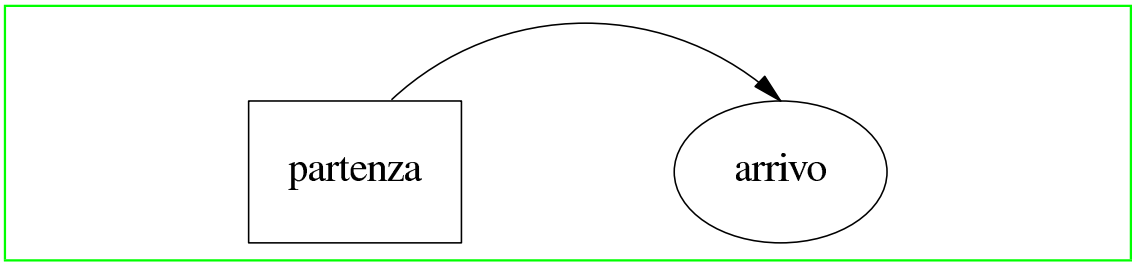
Il modificatore '**to**' può essere utilizzato da solo per specificare la destinazione di uno spostamento; analogamente per il modificatore '**from**'.

I modificatori '**from**' e '**to**' possono anche essere utilizzato assieme a '**line**' o '**arc**' per specificare gli estremi dell'oggetto. Usati assieme ai nomi di posizione, forniscono un meccanismo assai flessibile al fine di connettere i vari oggetti. Si considerino ad esempio il listato 5.46 e la corrispondente figura 5.47.

Listato 5.46.

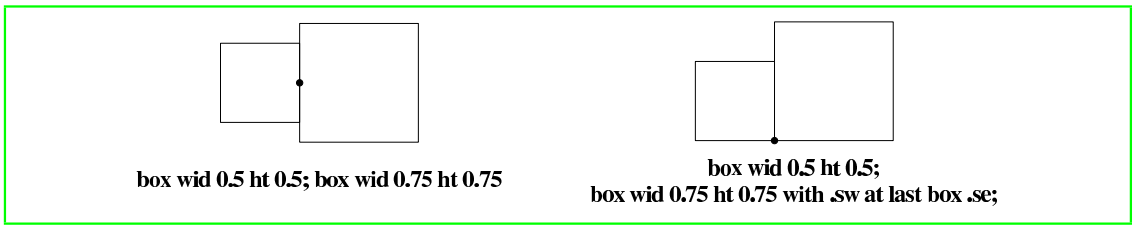
```
box "partenza"
move 0.75;
ellipse "arrivo"
arc -> cw from 1/3 of the way \
    between last box .n and last box .ne to last ellipse .n;
```

Figura 5.47. Una connessione un po' complicata, descritta con una sintassi naturale.



Il modifcatore '**with**' consente di identificare punti di due oggetti. Si tratta di un modo molto naturale per connettere gli oggetti. Si consideri, a mo' d'esempio, la figura 5.48.

Figura 5.48. Uso del modificatore '**with**' (sono evidenziati i punti di giunzione).



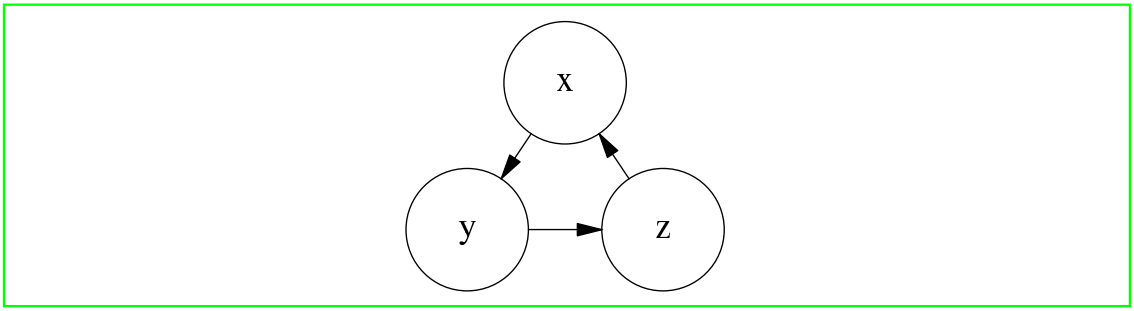
5.12.5 Il modificatore «chop»

Quando le linee tracciate fra due cerchi non li intersecano nei punti cardinali, è utile poterle accorciarle della lunghezza dei raggi dei cerchi ad entrambi i capi. Il listato 5.49 e la corrispondente figura 5.50 illustrano l'utlilizzo del modificatore '**chop**'.

Listato 5.49.

```
.PS
circle "x"
circle "y" at 1st circle - (0.4, 0.6)
circle "z" at 1st circle + (0.4, -0.6)
arrow from 1st circle to 2nd circle chop
arrow from 2nd circle to 3rd circle chop
arrow from 3rd circle to 1st circle chop
.PE
```

Figura 5.50. Il modificatore '**chop**'.



Si osservi che il modificatore '**chop**' sposta le punte di freccia, non le tronca. Per difetto, il modificatore '**chop**' accorcia entrambi i capi della linea di una lunghezza pari a '**circlerad**'. Aggiungendo un suffisso numerico è possibile modificare tale lunghezza.

Scrivendo:

```
line ... chop primo_raggio chop secondo_raggio
```

con ***primo_raggio*** e ***secondo_raggio*** valori numerici, è possibile modificare la lunghezza di accorciamento ad entrambi i capi. Usando tale descrizione assieme alle funzioni trigonometriche, è possibile scrivere codice che gestisce intersezioni ancora più complicate.

5.13 Gruppi di oggetti

Esistono due diverse maniere per raggruppare più oggetti in Pic:

- ***raggruppamento a graffe*** (*brace grouping*);
- ***blocco composto*** (*block composite*).

5.13.1 Raggruppamenti a graffe

Il metodo più semplice per raggruppare un insieme di oggetti è mediante una coppia di parentesi graffe ('**{**' e '**}**'). All'uscita dal raggruppamento la posizione corrente e la direzione corrente vengono reimpostate ai loro valori precedenti all'ingresso nel raggruppamento.

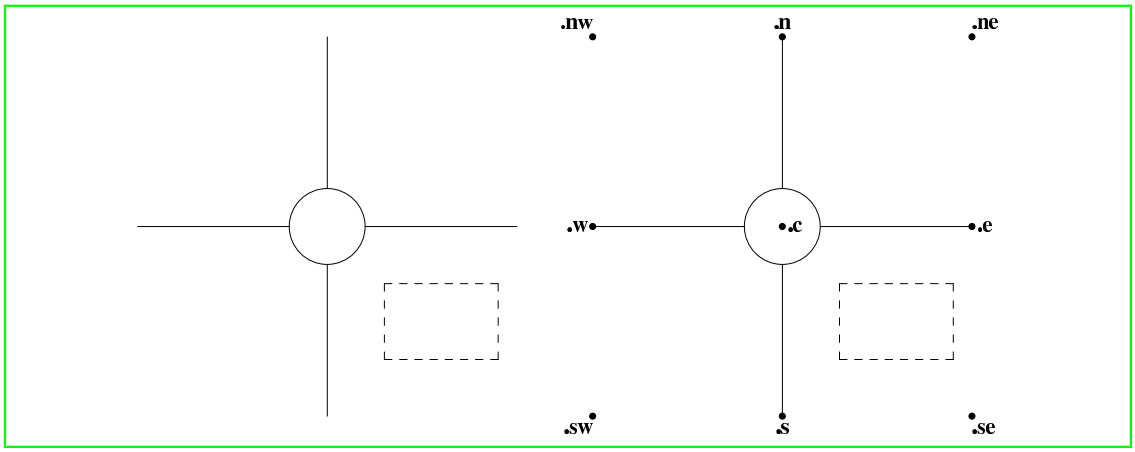
5.13.2 Blocchi composti

Un ***blocco composto*** è un oggetto creato da una successione di comandi racchiusa fra parentesi quadre (‘**[**’ e ‘**]**’). Il ***blocco composto*** è analogo per molti aspetti a un singolo oggetto chiuso, avente la forma e le dimensioni del riquadro circoscritto. Ad esempio, il frammento di codice del listato 5.51 corrisponde alla figura 5.52 (la quale mostra il blocco senza e con punti di giunzione.). L’etichetta ‘**A**’ corrisponde alla posizione del blocco.

Listato 5.51.

```
...
A: [
    circle;
    line up 1 at last circle .n;
    line down 1 at last circle .s;
    line right 1 at last circle .e;
    line left 1 at last circle .w;
    box dashed with .nw at last circle .se + (0.2, -0.2);
    Didascalia: center of last box;
]
...
```

Figura 5.52. Esempio di blocco composto.

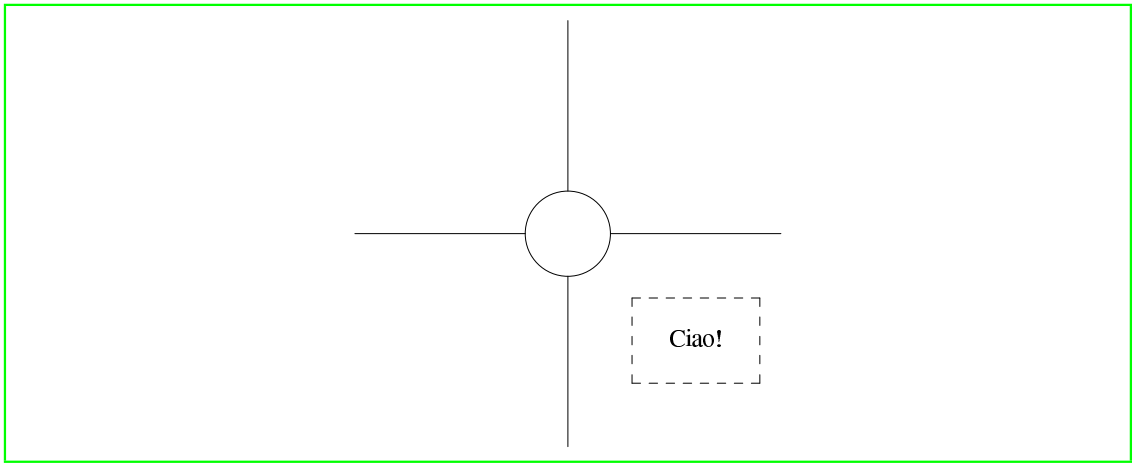


Per riferirsi a uno dei punti di giunzione del blocco composto, è possibile usare, ad esempio, la descrizione ‘**A .s**’. Ai fini della loro menzione, i blocchi composti sono delle ***classi***. Si può utilizzare la descrizione ‘**last [] .s**’ equivalentemente, ovunque sia necessaria l’indicazione di una posizione. Tale costrutto è molto importante al fine di assemblare diagrammi estesi e complessi.

I blocchi composti forniscono anche un meccanismo per il campo di azione delle variabili, analogamente a un ***ambiente*** Troff (v. sezione 5.23). Gli assegnamenti di variabile effettuati all’interno di un blocco vengono annullati all’uscita. Per accedere a un valore interno a un blocco, si deve scrivere il nome del blocco, seguito da un punto, seguito dall’etichetta desiderata. Ad esempio, si potrebbe far riferimento al centro del riquadro nel blocco composto, di cui a listato 5.51 e figura 5.52, mediante ‘**last [] .Didascalia**’ oppure ‘**A.Didascalia**’.

Questo tipo di riferimento a un’etichetta può essere utilizzato esattamente come una qualsiasi posizione. Ad esempio, aggiungendo ‘**"Ciao!" at A.Didascalia**’ al listato 5.51 si ottiene la figura 5.53.

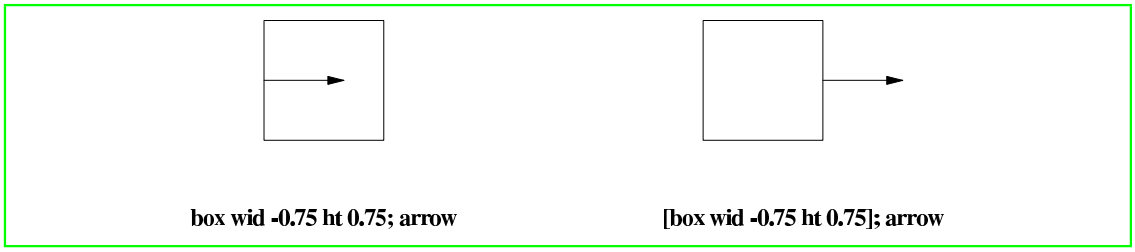
Figura 5.53. Aggiunta di una didascalia mediante un'etichetta interna.



È altresì possibile utilizzare le etichette interne sia a destra che a sinistra del modificatore '**`with`**'. Ciò significa che l'esempio di cui a listato 5.51 e figura 5.52 si potrebbe posizionare relativamente alla posizione del riquadro delle didascalia mediante un comando contentente '**`with A.Didascalia at`**'.

Si osservi che larghezza e altezza dei blocchi composti devono sempre essere considerate positive (figura 5.54).

Figura 5.54. I blocchi composti hanno sempre larghezza e altezza positive.



I blocchi composti si possono annidare; ciò significa che si possono utilizzare i punti di giunzione dei blocchi per costruire complesse strutture gerarchiche, dall'interno all'esterno. Si osservi che '**`last`**' e gli altri meccanismi di denominazione sequenziale non scendono ai livelli più interni, sicché nel listato 5.55 la freccia di cui alla riga 7 partirà dall'oggetto '**`P`**' e **non** dall'oggetto '**`Q`**'.

Listato 5.55. Blocchi composti annidati.

```
1  .PS
2  P: [box "foo"; ellipse "bar"];
3  Q: [
4       [box "baz"; ellipse "quxx"]
5       "testo assurdo";
6     ]
7  arrow from 2nd last [];
8  .PE
```

DWB Pic prevedeva che ci si potesse riferire ai livelli interni sino al massimo di un livello; GNU Gpic ha eliminato tale restrizione.

5.14 Variabili di stile

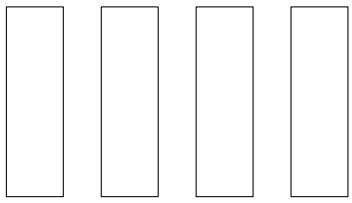
Esistono diverse variabili di stile globali in Pic, le quali possono essere utilizzate per variare il comportamento generale. Alcune di esse sono già state in precedenza menzionate; in questa sezione vengono presentate integralmente, specificandone anche il valore predefinito (tabella 5.56).

Tabella 5.56. Variabili di stile.

Variabile di stile	Valore predefinito (in pollici)	Che cosa determina
`moveht`	0,5	Ampiezza di uno spostamento vericale
`movewid`	0,75	Ampiezza di uno spostamento orizzontale
`lineht`	0,5	Ampiezza verticale di una linea (o spline) o di un segmento di spezzata
`linewid`	0,5	Ampiezza orizzontale di una linea (o spline) o di un segmento di spezzata
`textht`	0	Altezza del riquadro circoscritto a un oggetto di testo
`textwid`	0	Larghezza del riquadro circoscritto a un oggetto di testo
`arrowht`	0,1	Ampiezza longitudinale delle punte di freccia
`arrowwid`	0,05	Ampiezza trasversale delle punte di freccia
`arrowhead`	1	Abilita/disabilita campitura delle punte di freccia
`dashwid`	0,05	Intervallo di tratteggio per le linee
`maxpswid`	11	Larghezza massima della figura
`maxpsht`	8,5	Altezza massima della figura
`scale`	1	Fattore di scala
`fillval`	0,5	Valore di campitura

Tutte queste variabili possono essere impostate con una semplice istruzione di assegnamento (esempio in figura 5.57).

Figura 5.57.



boxht=1; boxwid=0.3; movewid=0.2; box; move; box; move; box; move; box;

In GNU Gpic impostando la variabile '**scale**' si scalano tutte le variabili di stile (dimensionali) in modo tale che i loro valori producano risultati equivalenti anche nella nuova scala. Per verificarlo, si consideri la seguente sessione interattiva (vedi anche sezione 5.67):

```
$ cat x.pic [ Invio ]

  .PS
  print boxwid
  oldboxwid=boxwid
  scale=2.54
  print boxwid
  print oldboxwid*scale
  .PE


$ cat x.pic | pic > /dev/null [ Invio ]

  0.75
  1.905
  1.905


$
```

Il comando '**reset**' reimposta tutte le variabili di stile al valore predefinito. Fornendo una lista di nomi di variabili come argomenti (opzionalmente separati da virgole) si possono selettivamente reimpostare solo alcune variabili. Lo stato delle variabili di stile persiste da figura a figura.

5.15 Espressioni, variabili e assegnamenti

Un numero è ovviamente un'espressione valida.[16] La notazione decimale è legittima; in GNU Gpic è anche legittima la notazione scientifica nello stile del linguaggio C (come '**5e-2**').

Ovunque si può utilizzare un numero, il linguaggio accetta anche una variabile. Le variabili possono essere variabili di stile predefinite (vedi sezione 5.14), oppure nuove variabili create da un assegnamento.

DWB Pic prevede solamente l'assegnamento ordinario mediante il simbolo di uguaglianza ('**=**'), il quale definisce la variabile (a primo membro) nel blocco corrente, a meno che non sia già stata ivi definita, e poi ne cambia il valore (che si trova a secondo membro) nel blocco corrente. La variabile non è visibile al di fuori del blocco.[17]

GNU Gpic prevede anche un tipo di assegnamento alternativo, mediante l'operatore '**:=**'. In tal caso la variabile **deve essere già stata definita**, e il valore viene assegnato alla variabile senza creare una variabile locale al blocco corrente. Si consideri la seguente sessione interattiva (vedi anche sezione 5.17.2):

```
$ cat x.pic [ Invio ]

  .PS
  x=5
  y=5
  [
    x:=3
    y=3
```

```
]
print x " " y
.PE
```

```
$ cat x.pic | pic > /dev/null [ Invio ]

  3 5

$
```

Nelle espressioni si possono utilizzare l'altezza, la larghezza, il raggio e le coordinate cartesiane di qualsiasi oggetto o vertice. Se '**A**' è un'etichetta associata a un oggetto, le espressioni elencate nel listato 5.62 sono tutte valide.

Listato 5.62. Espressioni valide a partire dall'etichetta '**A**'.

```
1  ...
2  A.x                  # ascissa del centro di A
3  A.ne.y               # ordinata del vertice nordest di A
4  A.wid                # larghezza di A
5  A.ht                 # la sua altezza
6  2nd last circle.rad  # raggio del penultimo cerchio
7  ...
```

Si noti in particolare l'espressione in riga 3 del listato 5.62, la quale mostra come estrarre le coordinate di un vertice.

Sono disponibili le espressioni aritmetiche fondamentali, con una sintassi simile al linguaggio C: '**+**', '*****', '**-**', '**/**' e '**%**'. Inoltre c'è '**^**' per l'elevamento a potenza. Si possono associare i termini in modo tradizionale mediante parentesi. GNU Gpic permette inoltre l'uso di operatori logici e di confronto: '**!**', '**&&**', '**||**', '**==**', '**!=**', '**>=**', '**<=**', '**>**', '**<**'.

Sono previste diverse funzioni predefinite: '**sin(*x*)**', '**cos(*x*)**', '**log(*x*)**', '**exp(*x*)**', '**sqrt(*x*)**', '**max(*x*,*y*)**', '**atan2(*x*,*y*)**', '**min(*x*,*y*)**', '**int(*x*)**', '**rand()**' e '**srand()**'. Sia '**exp**' che '**log**' si intendono in base 10; '**int**' produce il troncamento all'intero precedente; '**rand**' restituisce un numero pseudocasuale nell'intervallo [0..1) mentre '**srand**'[18] imposta il seme per una nuova successione di valori restituiti da '**rand**'.

GNU Gpic prevede anche una funzione ad un argomento '**rand(*x*)**', che restituisce un numero pseudocasuale nell'intervallo [0..***x***), ma si tratta di una funzione deprecata che potrebbe venir rimossa nelle future versioni del programma.

La funzione '**sprintf**' è simile all'omonima funzione della libreria standard del linguaggio C, ma prevede solamente le metavariabili di formato '**%**', '**%e**', '**%f**' e '**%g**'.

5.16 Macro

Pic offre la possibilità di definire delle ***macro***. La cosa può tornar utile per descrivere diagrammi ripetitivi. Assieme ale regole di visibilità per i blocchi composti (sezione 5.13.2) consente effettivamente di scrivere delle funzioni (vedi ad esempio il listato 5.63).

La sintassi è la seguente:

```
define nome_macro separatore corpo_macro separatore
```

In questo modo si definisce ***nome_macro*** come una macro che poi verrà sostituita dal testo in ***corpo_macro*** (separatori esclusi).[19] La macro può essere invocata come segue:

```
nome_macro(primo_argomento, secondo_primo_argomento, ...)
```

Gli (eventuali) argomenti vengono sostituiti al posto degli elementi sintattici ***$1***, ***$2***, ... ***$n*** che appaiono nel corpo della macro.

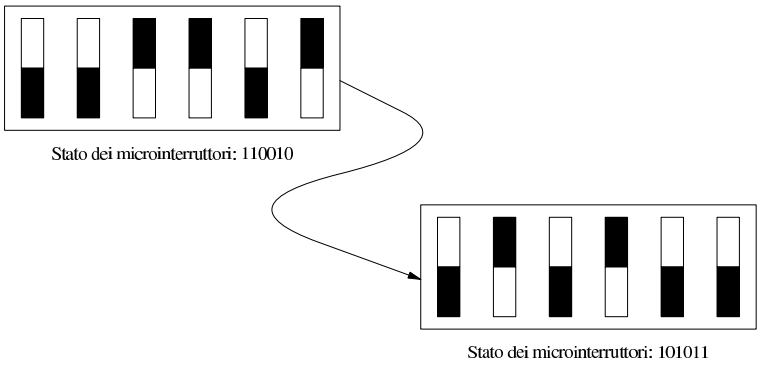
Come esempio di utilizzo delle macro, si considerino il listato 5.63 e la figura 5.64.

Listato 5.63. Esempio di utilizzo delle macro.

```
.PS
# Traccia un microinterruttore in un riquadro,
# $1 rappresenta lo stato (acceso/spento).
define jumper { [
    shrinkfactor = 0.8;
    Outer: box invis wid 0.45 ht 1;


    # Grazie al ] finale le seguenti variabili verranno
    # poi reimpostate automaticamente
    boxwid = Outer.wid * shrinkfactor / 2;
    boxht  = Outer.ht  * shrinkfactor / 2;


    box fill (!$1) with .s at center of Outer;
    box fill ($1)  with .n at center of Outer;
] }


# Traccia un blocco di sei microinterruttori
define jumperblock { [
    right;

    jumper($1);
    jumper($2);
    jumper($3);
    jumper($4);
    jumper($5);
    jumper($6);
```

```
    jwidth  = last [].Outer.wid;
    jheight = last [].Outer.ht;


    box with .nw at 6th last [].nw wid 6*jwidth ht jheight;


    "Stato dei microinterruttori: $1$2$3$4$5$6" at last box .s + (0, -0.2);
] }


# Esempio di invocazione macro.
A: jumperblock(1,1,0,0,1,0);
move right 2 then down 1;
B: jumperblock(1,0,1,0,1,1);
spline -> from A.right down right 1 then down left 2 to B.left
.PE
```

Figura 5.64. Esempio di utilizzo delle macro.



Un dettaglio che l'esempio del listato 5.63 non illustra è che il passaggio dei parametri alle macro non comporta la separazione delle stringhe: chiamando '**`jumper( 1 )`**', il valore di ***$1*** sarà '**`1`**'; chiamando '**`jumper(stringa lunga)`**' ***$1*** varrà '**`stringa lunga`**'.
Se si intende passare come parametro una coppia ordinata, per evitare conflitti con il simbolo di separazione degli argomenti è possibile avvolgere la coppia fra parentesi.

La definizione di una macro è persistente da una figura all'altra. Per annullare una definizione si scriva:

```
undef nome_macro;
```

per esempio, il codice del listato 5.65 annulla le due macro definite nel listato 5.63.

Listato 5.65.

```
...
undef jumper
undef jumperblock
...
```

5.17 Importazione ed esportazione dei dati

In questa sezione vengono descritti i comandi Pic che permettono lo scambio di dati fra Pic e l'ambiente esteno.

5.17.1 Inclusione di file e dati tabulati

Mediante la scrittura seguente:

```
copy nome_file
```

si inserisce il contenuto del file ***nome_file*** nel flusso di input di Pic. Le coppie di macro '**.PS**'/'**.PE**' eventualmente presenti vengono ignorate; in tal modo si possono includere figure precostruite.

Una variante[20] è la seguente:

```
copy [nome_file] thru {nome_macro | {separatore corpo_macro separatore}}
```

mediante la quale viene invocata la macro ***nome_macro*** - o eseguito il codice ***corpo_macro*** indicato in modo letterale - sugli argomenti ottenuti spezzando ciascuna riga del file ***nome_file*** in campi separati da spazi vuoti. La macro può avere sino a nove argomenti. Il codice ***corpo_macro*** dev'essere incluso in parentesi graffe, o fra una coppia di separatori (che però non possono comparire in ***corpo_macro***).

Se si omette di indicare il file, le righe da elaborare vengono prelevate dal resto del flusso di input di Pic, sino all'occorrenza successiva della macro '**.PE**'.

In ogni caso, assieme al comando '**copy**' GNU Gpic consente l'utilizzo di un suffisso '**until** ***parola***', la cui aggiunta permette di controllare la conclusione del comando '**copy**'[21].

Di conseguenza, i listati 5.66 e 5.67 si equivalgono.

Listato 5.66. '**copy**' con '**until**'.

```
.PS
copy thru % circle at ($1,$2) % until "FINE"
1 2
3 4
5 6
FINE
box
.PE
```

Listato 5.67. ‘**copy**’ senza ‘**until**’.

```
.PS
circle at (1,2)
circle at (3,4)
circle at (5,6)
box
.PE
```

5.17.2 Messaggi di *debug*

Il comando ‘**print**’ accetta un numero qualunque di argomenti, li concatena nella loro forma di output e invia il risultato verso lo standard error. Ciascun argomento può essere un’espressione, una posizione oppure una stringa di testo.

Per degli esempi elementari, si considerino le sezioni 5.14 e 5.15.

5.17.3 Invio di comandi al postprocessore

La seguente scrittura:

```
command argomento ...
```

fa sì che Pic concateni gli argomenti e li invii a Troff.[22] Ciascun ***argomento*** dev’essere un’espressione, una posizione o del testo.

Il funzionamento di ‘**command**’ è simile alle righe inizianti con ‘**.**’ (sezione 5.19) oppure con ‘****’ (sezione 5.20), con la differenza che permette di passare i valori delle variabili.

5.17.4 Esecuzione di comandi della shell

La scrittura seguente:

```
sh separatore testo_qualsiasi separatore
```

espande le macro eventualmante presenti in ***testo_qualsiasi***, dopodiché lo esegue come comando della shell.[23] Tale funzionalità può essere utilizzata per generare figure o file di dati per una successiva rielaborazione. I separatori possono essere una coppia bilanciata di parentesi graffe, oppure dei caratteri che non siano presenti in ***testo_qualsiasi***.[24]

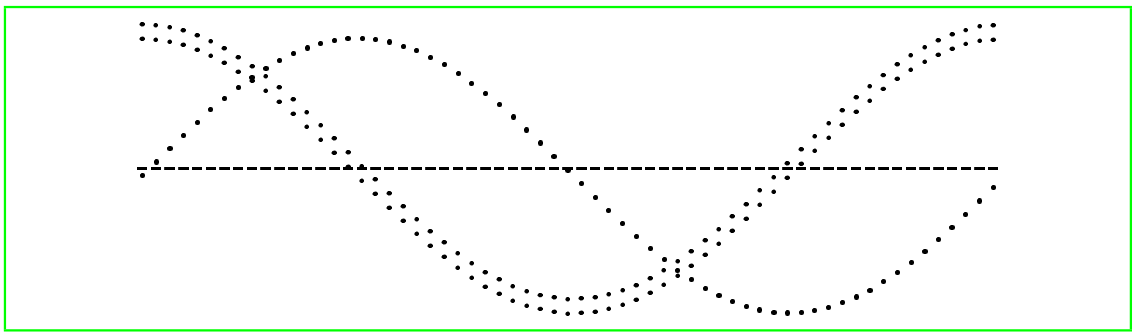
5.18 Controllo del flusso

Il linguaggio Pic prevede ***istruzioni condizionali*** e ***cicli enumerativi*** (listato 5.68 e figura 5.69).

Listato 5.68. Esempio di ciclo enumerativo.

```
.PS
pi = atan2(0, -1);
for i = 0 to 2 * pi by 0.1 do {
    "-" at (i/2, 0);
    "." at (i/2, sin(i)/2);
    ":" at (i/2, cos(i)/2);
}
.PE
```

Figura 5.69. Grafico realizzato mediante un ciclo enumerativo.



Ecco la sintassi per il ciclo enumerativo:

```
for variabile = prima_espressione to seconda_espressione [by [*]terza_espressione] ↩
↪do separatore corpo separatore
```

Ed eccone la semantica:

> Impostare ***variabile*** al valore di ***prima_espressione***. Fintantoché il valore di ***variabile*** non supera il valore di ***seconda_espressione***, eseguire ***corpo*** e incrementare ***variabile*** di una quantità pari a ***terza_espressione***; in assenza della parola chiave '**by**', incrementare ***variabile*** di una unità. Se ***terza_espressione*** è preceduta dal simbolo '*****' allora ***variabile*** deve venire moltiplicata per (**e non incrementata di**) una quantità pari a ***terza_espressione***.[25]

Ecco la sintassi per l'istruzione condizionale:

```
if espressione then separatore se_vero separatore [else altro_separatore se_falso altro_separatore]
```

Ed eccone la semantica:

> Valutare ***espressione***; se non è nulla allora eseguire ***se_vero***, altrimenti eseguire ***se_falso***.[26]

Nelle espressioni condizionali possono comparire gli usuali operatori relazionali: '**!**', '**&&**', '**||**', '**==**', '**!=**', '**>=**', '**<=**', '**>**', '**<**'.

È altresì previsto il confronto fra stringhe, mediante '**==**' e '**!=**'.

I confronti fra stringhe vanno inseriti fra parentesi per evitare ambiguità sintattiche.

5.19 Interfaccia verso *roff

L'output di Pic è costituito da comandi grafici per *roff.[27]

5.19.1 Argomenti per variare la scala

DWB Pic accetta uno oppure due argomenti di seguito alla macro '**.PS**'; essi vengono interpretati come larghezza e altezza[28] a cui va scalato il risultato dell'elaborazione di Pic.

GNU Gpic è meno flessibile: accetta solamente la larghezza finale a cui scalare l'immagine, oppure uno zero seguito dall'altezza massima a cui scalare; con due argomenti non nulli scalerà all'altezza massima.

5.19.2 Gestione della variazione di scala

Quando Pic elabora una descrizione proveniente dall'input, poi passa '**.PS**' e '**.PE**' al postprocessore. A '**.PS**' vengono aggiunti un paio di valori numerici, ossia le dimensioni della figura.[29] Il postprocessore utilizza tali valori per riservare abbastanza spazio per la figura e centrarla.

La realizzazione GNU del pacchetto di macro '**s**', ad esempio, include la definizione di cui al listato 5.70.

Listato 5.70. La definizione della macro '**.PS**' nel pacchetto di macro Groff '**s**'.

```
.\" ****************************
.\" ******** module pic ********
.\" ****************************
.\" Pic support.
.\" PS height width
.de PS
.br
.sp \\n[DD]u
.ie \\n[.$]<2 .@error bad arguments to PS (not preprocessed with pic?)
.el \{\
.       ds@need (u;\\$1)+1v
.       in +(u;\\n[.l]-\\n[.i]-\\$2/2>?0)
.\}
.HTML-IMAGE
..
```

C'è una definizione equivalente nel pacchetto di macro '**pic**', utilizzabile mediante l'opzione '**-mpic**'; ciò permette di usare anche pacchetti diversi da '**s**'.

Se si utilizza '**.PF**' al posto di '**.PE**', Troff reimposta la posizione allo stato precedente all'inizio

della figura.[30]

La scrittura seguente:

```
.PS <file
```

fa sì che il contenuto di ***file*** vada a sostituire la riga stessa.

Trattasi di una funzionalità deprecata. Si consiglia, invece, di utilizzare '**copy** ***file***' (v. sezione 5.17.1).

Per difetto, le righe in input che iniziano con un punto ('**.**') vengono passate al postprocessore, nella corrispondente posizione dell'output.

«Giocare» con le spaziature, sia orizzontali che verticali, è una probabile fonte di errori, mentre cambiare le dimensioni dei punti e le fonti di solito non comporta problemi. Analogamente è di solito innocuo un cambiamento di dimensioni o di fonte nelle stringhe, a patto di ripristinarle prima del termine della stringa.

Lo stato della campitura per *roff è persistente da figura a figura.

5.20 Interfaccia verso TeX

La modalità TeX è attivata dall'opzione '**-t**'. In tale modalità, Pic definisce, per ciascuna figura, una scatola verticale (v. sezione 5.23) denominata '**\graph**'. È responsabilità del codice TeX visualizzare effettivamente la scatola verticale, ad esempio con il comando '**\centerline{\box\graph}**'.

In effetti, poiché la scatola verticale ha altezza zero, il precedente comando produrrà uno spazio leggermente superiore sopra la figura rispetto a sotto; per evitarlo basta scrivere '**\centerline{\raise 1em\box\graph}**'.

Le righe che iniziano con il carattere barra obliqua inversa ('\') vengono passate intatte a TeX; viene automaticamente aggiunto un '**%**' alla fine di ciascuna riga al fine di evitare spazi indesiderati. Si può utilizzare tranquillamante tale funzionalità per cambiare le fonti, o per cambiare il valore di '**\baselineskip**'.

Qualsiasi altro utilizzo potrebbe generare risultati imprevisti. Usare con cautela!

Le righe che iniziano con un punto ('**.**') non subiscono nessuna elaborazione particolare.

La modalità TeX di Pic **non** traduce i cambiamenti di fonte e dimensione dei caratteri contenute nelle stringhe di testo!

5.21 Comandi obsoleti

GNU Gpic prevede il comando seguente:

```
plot espressione ["testo"]
```

Trattasi di un oggetto costruito utilizzando ***testo*** come stringa di formattazione (compatibile con '**`sprintf`**'), con argomento ***espressione***. Omettendo ***testo*** viene usata implicitamente la stringa '**`%g`**'. Si possono specificate degli attributi alla maniera di una normale stringa di testo.

Si presti particolare cautela alla stringa di formattazione usata, poiché Pic non fa controlli molto accurati al riguardo.
Il comando '**`plot`**' è deprecato; si consiglia di utilizzare '**`sprintf`**' al suo posto.

5.22 Indicazioni operative per ottenere le immagini

Per effettuare delle prove con il linguaggio Pic è necessario poter ottenere in qualche modo concreto l'immagine finale e visualizzarla. A tale scopo si indicano qui alcune possibilità.

Si può ad esempio utilizzare una pipeline del genere:

```
cat sorgente_pic | pic | troff -Tps | grops | ps2eps > file_eps
```

ottenendo quindi un file EPS, da visualizzare o ulteriormente elaborare. Addirittura si potrebbe preparare una riga di comando più complessa, la quale racchiuda in sé l'intero ciclo di modifica, compilazione e visualizzazione:

```
vi sorgente_pic; cat sorgente_pic | pic | troff -Tps | grops | ps2eps > file_eps; ↩
↪gv file_eps
```

Se la sintassi del sorgente Pic non è corretta, si ottiene un messaggio di errore standard, secondo lo schema seguente:

```
pic:sorgente_pic:numero_di_riga: syntax error before elemento
pic:sorgente_pic:numero_di_riga: giving up on this picture
```

ove ***elemento*** è un elemento sintattico, di solito immediatamente successivo alla causa dell'errore.

Un'alternativa sintatticamente più semplice prevede l'utlizzo di Pic2plot (parte del pacchetto GNU Plotutils):

```
cat sorgente_pic | pic2plot -Tps > file_eps
```

tenendo però presente che:

- Pic2plot riconosce solo un sottoinsieme del linguaggio Pic;
- Pic2plot non riconosce le sequenze di escape che iniziano con '**\(**', pertanto se si desiderano ottenere le lettere accentate si devono utilizzare delle sequenze di escape che iniziano con '\';
- l'aspetto finale dell'immagine può essere leggermente differente rispetto a quanto ottenuto mediante Troff.

Un'ulteriore semplificazione si può ottenere utilizzando il programma di Eric S. Raymond '**pic2graph**', il quale si appoggia a Groff e perciò riconosce l'intero linguaggio Pic:

```
cat sorgente_pic | pic2graph > file_png
```

Va precisato che, purtroppo, al momento della stesura del presente lavoro il programma '**pic2graph**' non funziona correttamente, nel senso che il file prodotto non ha le dimensioni corrette ed è pertanto inutilizzabile.

Se si intende utilizzare del codice Eqn all'interno del codice Pic, si tenga presente che (secondo quanto dichiarato nella pagina di manuale[31]) la realizzazione GNU di Eqn (Geqn) non è del tutto compatibile con il Troff tradizionale, pertanto è necessario utilizzare Groff:

```
cat sorgente | groff -e -p | ps2eps > file_eps
```

Per poter utilizzare del codice Eqn all'interno del codice Pic è necessario che nel sorgente si specifichino i delimitatori per il codice Eqn, mediante le macro '**.EQ**' e '**.EN**', ad esempio:

```
.EQ
delim $$
.EN
.PS
arrow; box "$1 over H(z)$"; arrow
.PE
```

Infine, si noti che Alml (il sistema di composizione SGML di Daniele Giacomini) offre supporto - a partire dall'estate 2005 - per il linguaggio Pic (vedi capitolo 4).

5.23 Riferimenti

- B. W. Kernighan, *PIC - A Graphics Language for Typesetting (Revised User Manual)*, Bell Labs Computing Science Technical Report #116, maggio 1991

  ⟨*http://cm.bell-labs.com/cm/cs/cstr/116.ps.gz* ⟩
- Eric S. Raymond, *Making Pictures With GNU PIC*[32]

  ⟨*http://www.catb.org/~esr/writings/taoup/html/graphics/pic.ps* ⟩

- Dale Dougherty, Tim O'Reilly, *UNIX ® Text Processing*, Hayden Books, edizione per Internet «UTP Revival», 2004

  ⟨*http://home.alltel.net/kollar/utp/*⟩

- Daniele Giacomini, *Appunti di informatica libera*, capitoli *Introduzione a *roff* e *TeX: paragrafi, righe, spazi, scatole e linee*

  ⟨*http://a2.swlibero.org/introduzione_a_roff.htm*⟩

  ⟨*http://a2.swlibero.org/tex_paragrafi_righe_spazi_scatole_e_linee.htm*⟩

---

[1] C'è poi il caso di coloro che, essendo disabili, hanno un impedimento fisico all'azione stessa del disegnare.

[2] **GNU Groff** GNU GPL

[3] ⟨*http://en.wikipedia.org/wiki/Petri_net*⟩

[4] ⟨*http://en.wikipedia.org/wiki/Concept_map*⟩

[5] Si tenga presente che il cambiamento di scala ha effetto solamente sui valori predefiniti delle variabili di stile; v. anche sezione 5.14.

[6] Purtroppo, la versione di GNU Gpic provata al momento della presente stesura (parte del pacchetto Groff nella versione 1.18.1-15) non sembra consentire la decorazione delle ellissi.

[7] Le punte di freccia solide vengono sempre campite con il colore di contorno corrente (v. sezione 5.8.8).

[8] situazione predefinita utilizzando Groff

[9] Ovviamente l'attributo non ha effetto per gli oggetti in cui i due punti coincidono.

[10] '**'th**' è un singolo elemento sintattico: non ci deve essere uno spazio fra '**'** e '**th**'.

[11] Si può considerare l'elemento sintattico '**:**' come una spece di operatore di assegnamento.

[12] Espresse i pollici.

[13] Per la precisione, possono essere addizionate una posizione e una coppia di espressioni; quest'ultima deve comparire come secondo termine dell'operazione.

[14] Un possibile utilizzo è per posizionare correttamente delle didascalie.

[15] I valori **non** sono ristretti all'intervallo [0..1].

[16] Tutti i numeri sono conservati internamente come in virgola mobile

[17] La situazione è simile al linguaggio C, in cui una variabile in un blocco impedisce di accedere alla variabile omonima esterna al blocco.

[18] Estensione GNU.

[19] ***separatore*** può essere qualunque carattere che non compaia in ***corpo_macro***, oppure la coppia ***separatore***/***separatore*** può essere costituita da una coppia bilanciata di parentesi graffe ('**{**' e '**}**'), e (in ogni caso) ***corpo_macro*** può contenere coppie bilanciate di parentesi graffe.

[20] Mututata dal linguaggio Grap.

[21] In pratica, il comportamento predefinito equivale a '**`until .PE`**'.

[22] Oppure a TeX, se vengono utilizzate le opzioni '**`-t`**' o '**`-c`**'.

[23] Per difetto, GNU Gpic non esegue in realtà il comando, per motivi di sicurezza. Per forzare l'esecuzione bisogna invocare il programma con l'opzione '**`-U`**'.

[24] In entrambi i casi, ***testo_qualsiasi*** può contenere coppie bilanciate di parentesi graffe, e le eventuali stringhe possono contenere anche graffe non bilanciate.

[25] ***separatore*** può essere qualsiasi carattere che non compaia in ***corpo*** (o, alternativamente, la coppia ***separatore***/***separatore*** può essere una coppia bilanciata di parentesi graffe, come nel caso del comando '**`sh`**', v. sezione 5.17.4).

[26] ***separatore*** può essere qualsiasi carattere che non compaia in ***se_vero***. ***altro_separatore*** può essere qualsiasi carattere che non compaia in ***se_falso***. Entrambe le coppie di separatori possono in alternativa essere una coppia bilanciata di parentesi graffe, come nel caso del comando '**`sh`**' (v. sezione 5.17.4). In ogni caso sia ***se_vero*** sia ***se_falso*** possono contenere coppie bilanciate di parentesi graffe. Il bilanciamento non è richiesto per parentesi graffe presenti nelle stringhe di testo.

[27] GNU Gpic si fonda su estensioni di tipo grafico presenti in Groff ma non in Troff.

[28] In pollici. Le dimensioni sono indipendenti.

[29] In pollici.

[30] Kernighan (sezione 5.23) fa notare che la lettera «F» sta per *flyback*.

[31] *«[...] The syntax is quite compatible with Unix eqn. The output of GNU eqn cannot be processed with Unix troff; it must be processed with GNU troff. [...]»*

[32] A questo lavoro si deve gran parte dei contenuti del presente capitolo.

Grafici e diagrammi con Jgraph

Jgraph[1] è un programma che serve a tracciare grafici e diagrammi strutturati. Diversamente da altri strumenti per il disegno, Jgraph è un programma **non interattivo**: legge un file di input (il ***sorgente***) e produce un file di output (la ***presentazione***) in formato PostScript oppure EPS. L'output può in seguito essere quindi visualizzato, stampato, incorporato in un altro file per ulteriori elaborazioni oppure convertito in altri formati.

Inoltre, si noti che Alml (il sistema di composizione SGML di Daniele Giacomini) offre supporto - a partire da gennaio 2006 - per il linguaggio Jgraph (vedi capitolo 4).

Questo capitolo costituisce una introduzione di livello elementare all'uso e alle caratteristiche principali del programma, senza alcuna prestesa di essere esauriente; la documentazione di riferimento più completa per Jgraph è costituita dalla pagina di manuale, cui si rinvia il lettore interessato per gli eventuali approfondimenti.

6.1 Invocazione

Ecco l'utilizzo tipico di Jgraph:

```
$ jgraph in.jgr > out.eps [ Invio ]
```

oppure:

```
$ jgraph -P in.jgr > out.ps [ Invio ]
```

o, ancora:

```
$ cat in.jgr | jgraph -P | ps2eps -f -l -q > out.eps [ Invio ]
```

e simili.

In assenza di un nome di file dato come argomento, Jgraph legge dallo standard input.

6.2 Esempi elementari

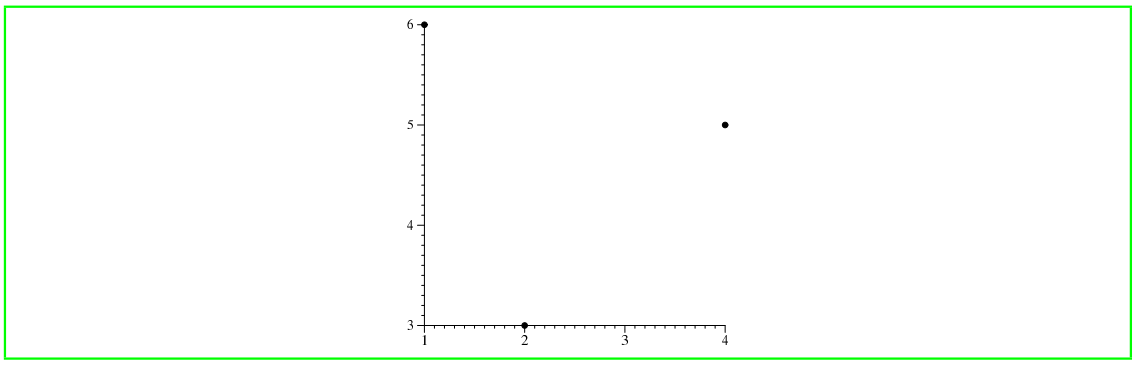
Al fine di prendere confidenza con il programma Jgraph, cominciamo vedere alcuni semplici esempi. Per indicare a Jgraph di tracciare un nuovo grafico o diagramma si utilizza la parola chiave '**newgraph**'; quindi si aggiungono delle curve al diagramma mediante '**newcurve**'; infine si aggiungono dei punti alla curva mediante '**pts**'.

Il listato 6.1 mostra un esempio elementare, presentato graficamente in figura 6.2. Si tratta semplicemente di disegnare tre punti: (2,3), (4,5) e (1,6). Jgraph si occupa di aggiungere gli assi di riferimento e di scegliere lo stile di tracciamento dei punti; è un esempio decisamente banale, tanto per iniziare.

Listato 6.1. Jgraph: un esempio elementare.

```
(* Simple jgraph *)

newgraph

newcurve pts 2 3   4 5   1 6
```

Figura 6.2.



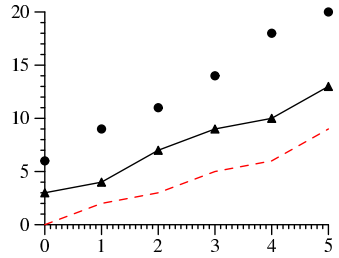
Si considerino adesso il listato 6.3 e la corrispondente figura 6.4. Si tratta di un esempio leggermente più complesso: nelle righe 1-3 si dà inizio al grafico mediante '**newgraph**', quindi si definiscono le dimensioni degli assi;[2] le righe 5-9 tracciano tre curve: nella prima si lascia che sia Jgraph a scegliere lo stile di tracciamento, nella seconda si specifica di indicare i punti mediante triangoli connessi da una linea solida, nella terza si specifica di non indicare i punti ma di tracciare solamente la linea che li unisce (la linea sarà tratteggiata e di colore rosso[3]). Si noti che l'intervallo visibile degli assi è stato automaticamente calcolato in modo da contenere precisamente tutti i punti specificati nel sorgente.

Nella pagina di manuale si trovano definiti i possibili stili da associare allle parole chiave '**marktype**' e '**linetype**'. Si noti in particolare che '**newline**' è semplicemente un sinonimo per '**newcurve marktype none linetype solid**'.

Listato 6.3. Jgraph: un grafico con tre curve di tipo differente.

```
1  newgraph
2  xaxis size 2
3  yaxis size 1.5
4
5  newcurve pts 0 6  1 9  2 11  3 14  4 18 5 20
6  newcurve marktype triangle linetype solid
7           pts 0 3  1 4  2 7  3 9  4 10 5 13
8  newcurve marktype none linetype dashed color 1 0 0
9           pts 0 0  1 2  2 3  3 5  4 6  5 9
```

Figura 6.4.



6.3 Alcuni dettagli sulla sintassi

L'algoritmo di riconoscimento dell'input da parte di Jgraph è basato non sulle righe bensì sugli elementi sintattici (*token-based*); Jgraph semplicemente lavora su parole separate da spazi bianchi; i commenti devono essere inclusi fra '**`(*`**' e '**`*)`**'. Gli oggetti fondamentali per Jgraph sono:

- la ***pagina***;
- i ***grafici***;
- gli ***assi***;
- le ***curve***;
- le ***stringhe***;
- le ***legende***.

Si può pensare che, quando si opera sul sorgente, si stanno effettivamente creando o modificando tali oggetti nella presentazione. Pertanto, quando si scrive '**`newcurve`**', si sta in effetti modificando o creando una curva, per la quale è possibile specificare ***attributi***, come '**`marktype`**' e '**`linetype`**', e dei punti. La maggior parte degli attributi possiedono un valore predefinito che però è possibile cambiare. Se si specifica un attributo più di una volta Jgraph considera valida l'ultima indicazione, sicché ad esempio '**`newcurve marktype box marktype circle`**' equivale a '**`newcurve marktype circle`**'.

In parziale deroga a quanto or ora specificato, la parola chiave '**`pts`**' agisce in maniera «additiva», pertanto '**`newcurve pts 0 0 1 1 2 2`**' equivale a '**`newcurve pts 0 0 pts 1 1 pts 2 2`**'.

È inoltre possibile includere dei file esterni nel sorgente Jgraph mediante '**`include`** ***nome_file***'.

6.4 Gestione degli assi

Se capita di non gradire la maniera in cui Jgraph produce automaticamente gli assi, può tornare utile la tabella 6.5.

Tabella 6.5. Jgraph: alcuni attributi degli assi.

Attributo ed eventuali valori	Significato
`size` ***dimensione***	Imposta la dimensione dell'asse a ***dimensione*** (in pollici).
`min` ***valore***	Imposta il valore minimo a ***valore***.
`max` ***valore***	Imposta il valore massimo a ***valore***.
`hash` ***numero_valori***	Traccia una tacca (primaria) e un'etichetta di tacca ogni ***numero_valori*** valori.
`mhash` ***numero_tacche***	Traccia ***numero_tacche*** tacche secondarie fra ogni due tacche primarie consecutive.
`gray` ***scala_di_grigio***	Imposta la scala di grigio dell'asse a ***scala_di_grigio***, ove zero significa nero e uno significa bianco.
`color` ***componente_rossa*** ↩ ↪ ***componente_verde*** ↩ ↪ ***componente_azzurra***	Imposta il colore dell'asse, codificato mediante la terna RGB indicata.
`nodraw`	Impedisce il tracciamento dell'asse, in ogni sua parte.
`draw`	Garantisce il tracciamento dell'asse, in ogni sua parte.
`log`	Imposta la scala dell'asse come logaritmica.
`linear`	Imposta la scala dell'asse come lineare.
`no_draw_hash_marks`	Impedisce il tracciamento delle tacche lungo l'asse.
`no_draw_hash_labels`	Impedisce il tracciamento delle etichette lungo l'asse.
`draw_at` ***valore***	Traccia l'asse in una posizione diversa dal valore minimo.
`label` ***etichetta***	Imposta l'etichetta dell'asse.

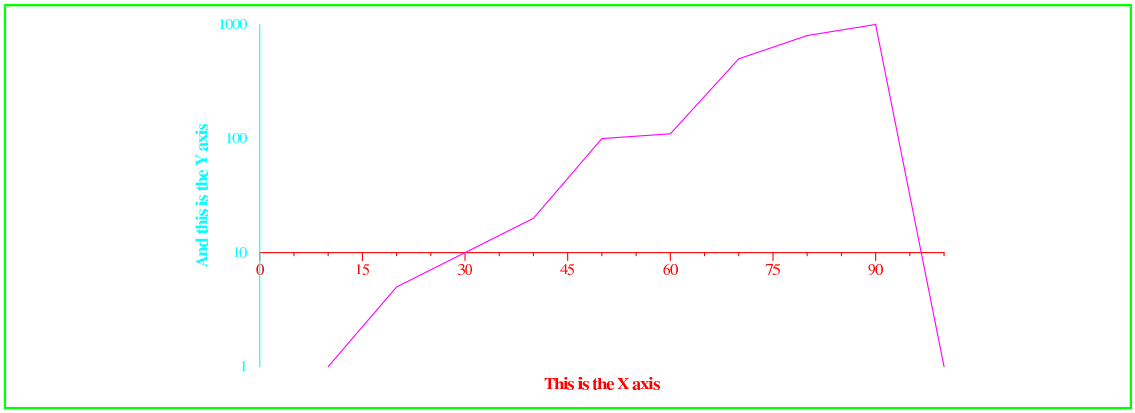
Il listato 6.6 e la figura 6.7 illustrano l'uso di alcuni degli attributi descritti nella tabella 6.5.

Listato 6.6. Jgraph: uso di alcuni degli attributi degli assi.

```
newgraph

xaxis
  size 6
  min 0 max 100
  hash 15 mhash 2 (* i.e. minor hashes at the 5's and 10's *)
  color 1 0 0
  label : This is the X axis
  draw_at 10

yaxis
```

```
   size 3
   min 1 max 1000
   log
   no_draw_hash_marks
   label : And this is the Y axis
   color 0 1 1

newline color 1 0 1
   pts 10 1
       20 5
       30 10
       40 20
       50 100
       60 110
       70 500
       80 800
       90 1000
       100 1
```

Figura 6.7.



6.5 Gestione delle curve

Jgraph assume dei valori predefiniti per le dimensioni degli ***indicatori di punto***, ma è possibile impostarli diversamente mediante '**`marksize`** ***larghezza altezza***'. La larghezza si intende nelle unità dell'***asse delle ascisse***, a meno che la scala non sia logaritmica nel qual caso si intende espressa in pollici; analogamente l'altezza fa riferimento all'unità dell'***asse delle ordinate***.

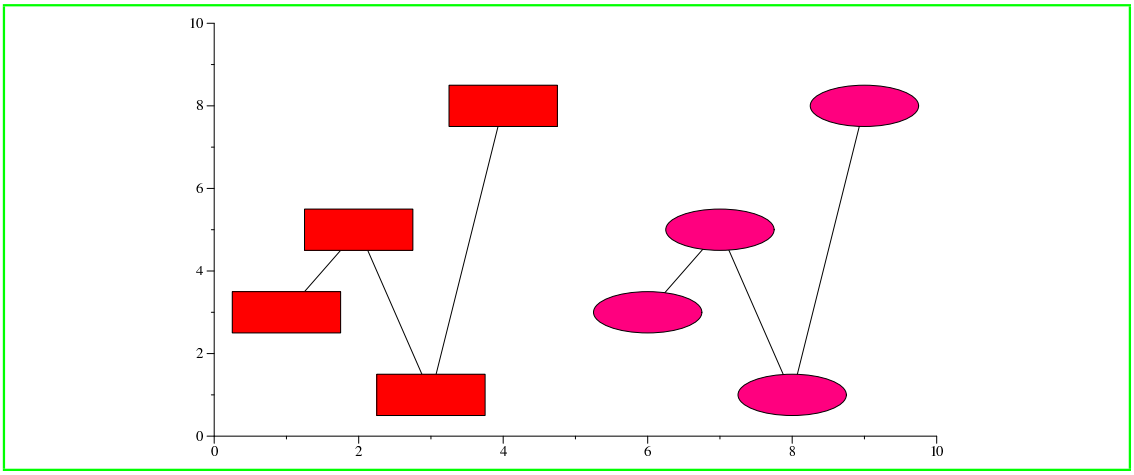
La parola chiave '**`copycurve`**' permette di creare una curva avente i medesimi attributi di quella precedente (ma non gli stessi punti, ovviamente).

Il listato 6.8 e la figura 6.9 illustrano un esempio. Si noti che '**`cfill`**' riempie l'interno degli indicatori con il colore specificato.

Listato 6.8. Jgraph: esempio di gestione delle curve.

```
newgraph

xaxis min 0 max 10 size 7
yaxis min 0 max 10 size 4

newcurve marktype box marksize 1.5 1 linetype solid cfill 1 0 0
   pts 1 3  2 5  3 1  4 8

copycurve marktype ellipse cfill 1 0 .5
   pts 6 3  7 5  8 1  9 8
```

Figura 6.9.



6.6 Stringhe

La gestione delle stringhe in Jgraph avviene mediante la parola chiave '**`newstring`**'. Gli attributi delle stringhe sono la ***posizione***, la ***fonte***, la ***dimensione della fonte***, il ***colore***, la ***rotazione***, l'***allineamento*** e il ***testo***. La posizione si imposta mediante '**`x`** ***posizione_x*** **`y`** ***posizione_y***'. La fonte dev'assere una fonte PostScript. Le fonti standard sono '**`Times-Roman`**', '**`Times-Italic`**', '**`Times-Bold`**', '**`Helvetica`**', '**`Helvetica-Italic`**', '**`Helvetica-Bold`**', '**`Courier`**' e '**`Symbol`**'. Il colore viene indicato tramite '**`lcolor`**', oppure si utilizza '**`lgray`**' per indicare la scala di grigio. Gli allineamenti si specificano come segue:

- '**`hjl`**', '**`hjr`**' e '**`hjc`**' corrispondono a sinistra, destra e centro (orizzontale) rispettivamente;
- '**`vjt`**', '**`vjb`**' e '**`vjc`**' corrispondono a sopra, sotto e centro (verticale) rispettivamente.

Si può ruotare una stringa mediante '**`rotate`** ***gradi***'. Si imposta il testo di una stringa mediante il carattere '**`:`**', seguito da uno spazio bianco e quindi dal testo vero e proprio. Per ottenere del testo su più righe è possibile utlizzare il carattere '**`\`**' al termine di ciascuna riga tranne l'ultima.

Anche le ***etichette*** dei grafici sono stringhe, quindi possono utilizzare gli attributi or ora descritti.

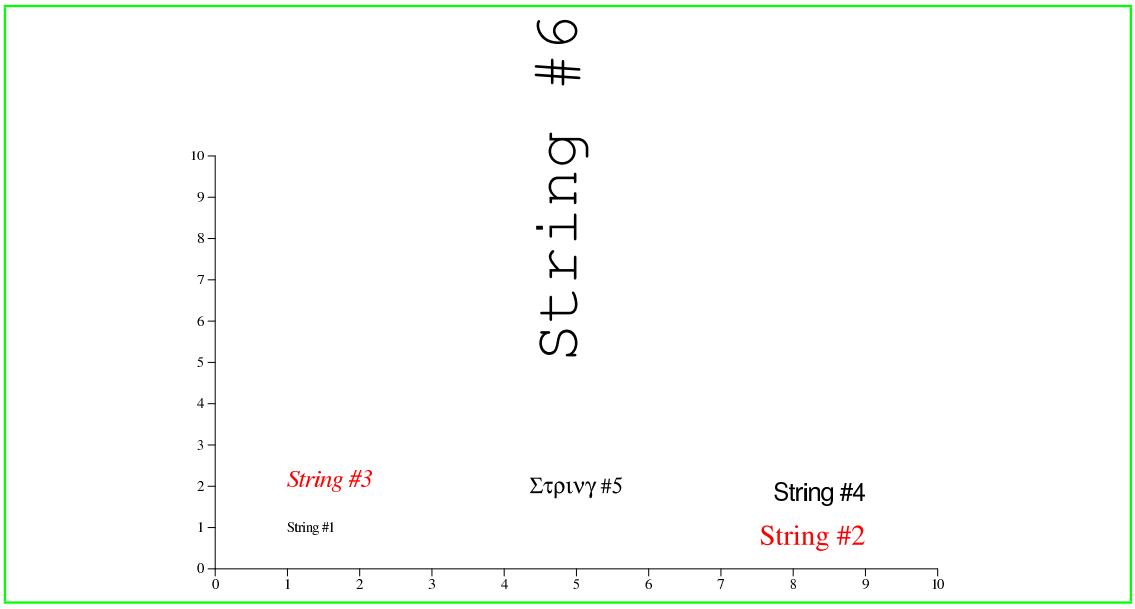
'**`copystring`**' copia tutti gli attributi di una stringa (testo incluso).

Il listato 6.10 e la figura 6.11 illustrano un esempio.

Listato 6.10. Jgraph: gestione delle stringhe.

```
newgraph
  xaxis min 0 max 10 hash 1 mhash 0 size 7
  yaxis min 0 max 10 hash 1 mhash 0 size 4

newstring hjl vjc x 1 y 1 : String #1

newstring hjr vjt x 9 y 1 fontsize 20 lcolor 1 0 0 : String #2

copystring hjl vjb x 1 y 2 fontsize 16 font Times-Italic : String #3

newstring hjr vjt x 9 y 2 fontsize 16 font Helvetica : String #4

newstring hjc vjc x 5 y 2 fontsize 16 font Symbol : String #5

newstring hjl vjb fontsize 45 font Courier rotate 90 x 5 y 5 : String #6
```

Figura 6.11.



6.7 Legende

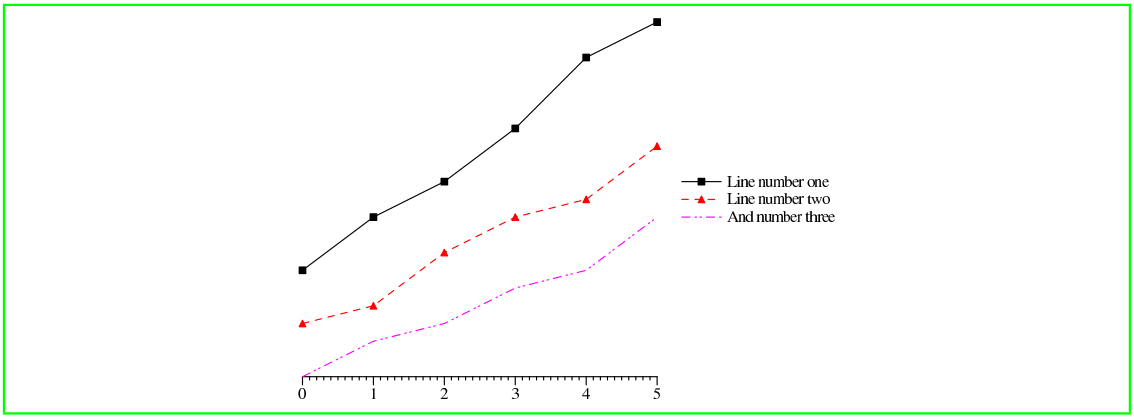
È possibile applicare un'etichetta a una curva mediante l'attributo '**`label`**': esso indica a Jgraph di creare una voce in ***legenda*** per la curva data. Il listato 6.12 e la figura 6.13 forniscono un esempio.

Listato 6.12. Jgraph: esempio di legenda.

```
newgraph
xaxis min 0 max 5
yaxis min 0 nodraw

newcurve marktype box linetype solid label : Line number one
   pts 0 6  1 9  2 11  3 14  4 18 5 20
newcurve marktype triangle linetype dashed color 1 0 0 label : Line number two
   pts 0 3  1 4  2 7  3 9  4 10 5 13
```

```
newcurve marktype none linetype dotdotdash color 1 0 1 label : And number three
        pts 0 0  1 2  2 3  3 5  4 6  5 9
```

Figura 6.13.

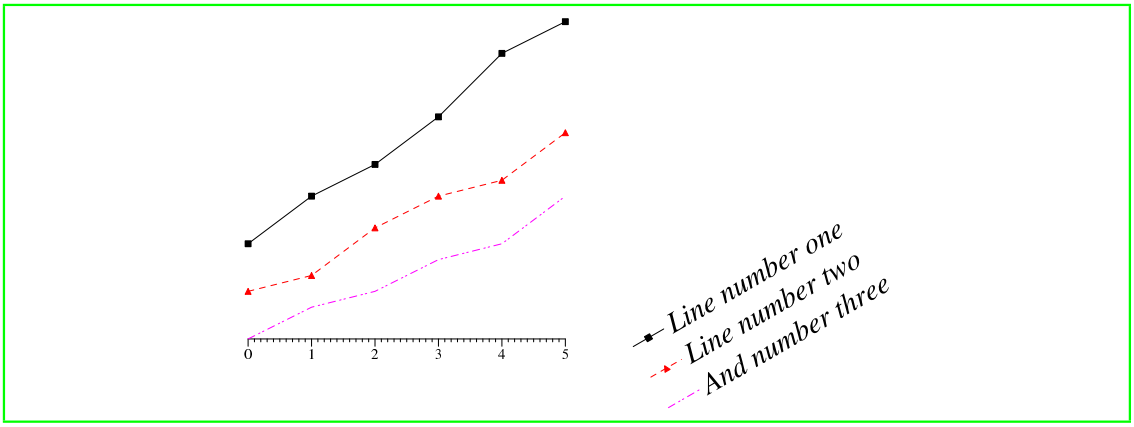


Si possono cambiare la fonte e la posizione della legenda, come unità complessiva, scrivendo '**`legend defaults`**' e specificando gli attributi per le stringhe: in tal modo si modificano tutte le stringhe della legenda. Il listato 6.14 e la figura 6.15 forniscono un esempio; è istruttivo provare a modificarlo e vedere quali effetti ne sortiscano. Le legende hanno altre interessanti caratteristiche, che si trovano ben descritte nella pagina di manuale.

Listato 6.14. Jgraph: un altro esempio di legenda.

```
newgraph
xaxis min 0 max 5
yaxis min 0 nodraw

newcurve marktype box linetype solid label : Line number one
   pts 0 6  1 9  2 11  3 14  4 18 5 20
newcurve marktype triangle linetype dashed color 1 0 0 label : Line number two
   pts 0 3  1 4  2 7  3 9  4 10 5 13
newcurve marktype none linetype dotdotdash color 1 0 1 label : And number three
        pts 0 0  1 2  2 3  3 5  4 6  5 9

legend defaults font Times-Italic fontsize 20 rotate 30 hjl vjt x 6 y 0
```

Figura 6.15.



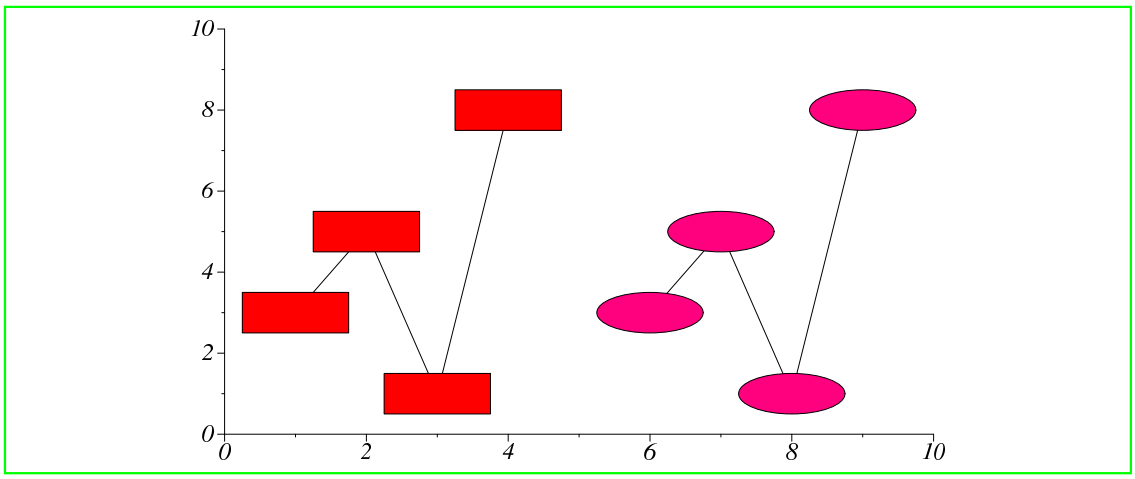
6.8 Etichette di tacca

Analogamente al caso delle legende è possibile modificare l'insieme delle ***etichette di tacca*** di un asse, come unità complessiva, mediante l'attributo '**`hash_labels`**': ad esempio, per cambiare le etichette di tacca in figura 6.9 in maniera tale che la fonte sia '**`Times-Italic`**' e la dimensione della fonte sia 16, si può scrivere '**`hash_labels font Times-Italic fontsize 16`**', e analogamente per l'asse delle ordinate (listato 6.16, righe 4 e 7, e figura 6.17).

Listato 6.16. Jgraph: gestione delle etichette di tacca.

```
1   newgraph
2
3   xaxis min 0 max 10 size 7
4     hash_labels font Times-Italic fontsize 16
5
6   yaxis min 0 max 10 size 4
7     hash_labels font Times-Italic fontsize 16
8
9   newcurve marktype box marksize 1.5 1 linetype solid cfill 1 0 0
10     pts 1 3  2 5  3 1  4 8
11
12  copycurve marktype ellipse cfill 1 0 .5
13     pts 6 3  7 5  8 1  9 8
```

Figura 6.17.



6.9 Grafici a barre

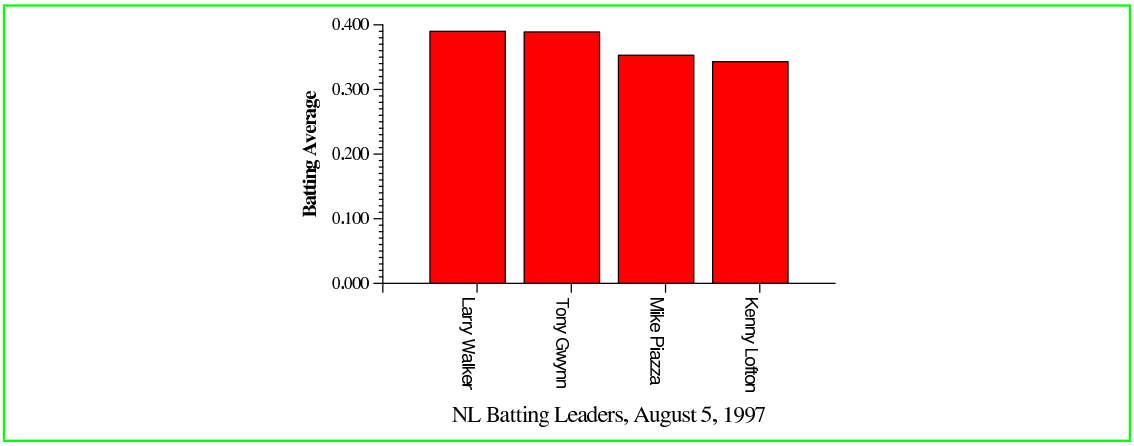
Utilizzando come indicatori '**`xbar`**' e '**`ybar`**' si ottengono i grafici a barre: '**`xbar`**' produce barre che si estendono parallelamente all'asse delle ordinate, '**`ybar`**' barre che si estendono parallelamente all'asse delle ascisse.

Il listato 6.18 e la figura 6.19 forniscono un interessante esempio di grafico a barre.

Listato 6.18. Jgraph: grafico a barre.

```
1   newgraph
2
3   xaxis
4     min 0.1 max 4.9
5     size 3.5
6     hash 1 mhash 0 no_auto_hash_labels
7
8   yaxis
9     min 0 max .4
10    size 2
11    precision 3
12
13  newcurve marktype xbar cfill 1 0 0 marksize .8
14    pts   1 .390
15          2 .389
16          3 .353
17          4 .343
18
19  xaxis
20    hash_label at 1 : Larry Walker
21    hash_label at 2 : Tony Gwynn
22    hash_label at 3 : Mike Piazza
23    hash_label at 4 : Kenny Lofton
24    hash_labels hjl vjc font Helvetica rotate -90
25
26  yaxis label : Batting Average
27  title : NL Batting Leaders, August 5, 1997
```

Figura 6.19.



Alcune osservazioni in merito al listato 6.18:

- **Righe 3-6**

  Si imposta la dimensione e l'intervallo dei valori per l'asse delle ascisse e si impostano le tacche sulle unità senza tacche secondarie; inoltre si indica a Jgraph di non generare le etichette di tacca in modo automatico, poiché si desidera personalizzarle.

- **Righe 8-11**

Niente di particolare da notare tranne l'attributo '**`precision`**' il quale specifica di includere tre decimali nelle etichette di tacca per l'asse delle ordinate.

- **Righe 13-17**

  Traccia la curva tramite barre rosse verticali.

- **Righe 19-24**

  Ulteriore modifica che interessa l'asse delle ascisse, generando etichette di tacca per ciascun punto e successivamente modificandole in maniera che appaiano ruotate di 90 gradi (in senso orario).

- **Righe 26-27**

  Si genera un etichetta per l'asse delle ordinate e un titolo per il grafico.

L'esempio illustrato nella presente sezione dovrebbe convincere delle possibilità del programma Jgraph: è possibile generare grafici anche di una certa complessità in maniera abbastanza naturale.

6.10 Particolarità

6.10.1 Poligoni

La possibilità di tracciare dei ***poligoni*** estende non poco le capacità grafiche del programma Jgraph: è sufficiente applicare l'attributo '**`poly`**' alla curva da tracciare. Gli attributi '**`pfill`**' e '**`pcfill`**' permettono inoltre di specificare una scala di grigio o un colore per la campitura del poligono. L'attributo '**`linethickness`**' consente di controllare lo spessore del perimetro.[4] È possibile campire il poligono con un disegno a strisce in alternativa alla campitura solida mediante '**`ppattern stripe`** ***inclinazione***', ove ***inclinazione*** controlla l'inclinazione delle strisce. Infine, se si indica `-1` come colore di campitura del poligono viene tracciato solamente il bordo mentre l'interno rimane vuoto, il che può essere talvolta desiderabile.

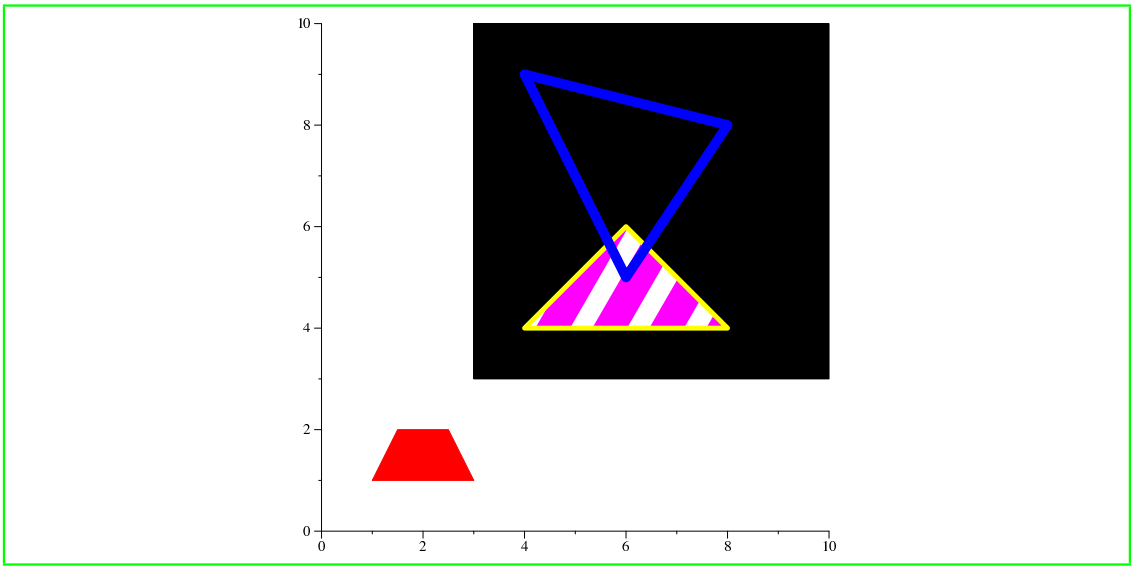
Il listato 6.20 e la figura 6.20 presentano un esempio con alcuni poligoni di diversa campitura. Si osservi che Jgraph traccia le curve nell'ordine indicato, sicché è possibile prevedere le eventuali sovrapposizioni.

Listato 6.20. Jgraph: poligoni.

```
newgraph

xaxis min 0 max 10 size 5
yaxis min 0 max 10 size 5

(* Draw a red trapezoid -- setting both the fill and color to be red means
   that the entire trapezoid, and not just the interior, will be red. *)

newline poly pcfill 1 0 0 color 1 0 0
   pts 1 1  3 1   2.5 2  1.5 2

(* Draw a big black square *)

newline poly pfill 0 pts 3 3  10 3  10 10  3 10
```

```
(* Draw a thick yellow triangle with a purple, striped interior inside the black
   square *)

newline poly linethickness 5 color 1 1 0 pcfill 1 0 1 ppattern stripe 60
   pts  4 4   8 4  6 6

(* Draw a blue triangle with a thick border no fill *)

newline poly linethickness 10 color 0 0 1 pfill -1 pts 4 9  6 5 8 8
```

Figura 6.21.



6.10.2 Curve di Bézier

Si possono generare delle linee curve con Jgraph se si interpretano alcuni dei punti indicati come ***punti di controllo*** per una ***curva di Bézier***.

Nella sezione 6.13 vengono indicati dei possibili spunti di approfondimento sulla questione, ma si consideri che per definire una curva di Bézier con Jgraph servono almeno quattro punti, e in generale è possibile tracciare una curva di Bézier mediante 3****n***+1 punti di controllo, di cui 2****n*** di controllo e i rimanenti ***n***+1 di passaggio della curva.[5]

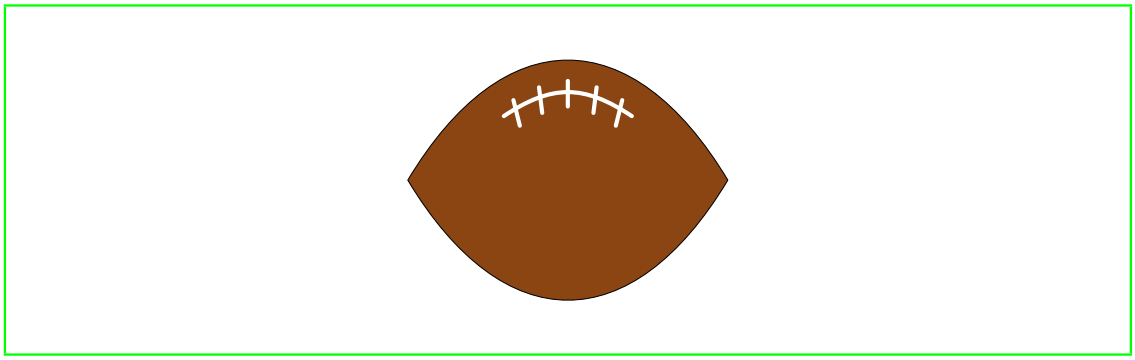
Il listato 6.22 e la figura 6.23 fornuscono un esempio in cui viene disegnato un pallone da football americano (molto stilizzato a dire il vero) mediante curve di Bézier.

Listato 6.22. Jgraph: curve di Bézier.

```
newgraph
xaxis min 0 max 10 nodraw
yaxis min 0 max 10 nodraw

(* Draw the outline of the football *)
newline bezier poly pcfill .543 .270 .074 pts
  0 5   3 10   7 10    10 5
        7 0    3 0     0 5

(* Draw the main thread *)
```

```
newline bezier linethickness 4 gray 1 pts
  3 7 4.5 8 5.5 8 7 7

(* Draw the crossthreads *)

copycurve nobezier pts 3.5 6.7 3.3 7.5
copycurve pts 6.5 6.7 6.7 7.5
copycurve pts 4.2 7.1 4.1 7.9
copycurve pts 5.8 7.1 5.9 7.9
copycurve pts 5 7.3  5 8.1
```

Figura 6.23.



6.11 Altre caratteristiche

Accenniamo ad alcune ulteriori interessanti caratteristiche del programma Jgraph, rinviando il lettore interessato alla sezione 6.13 per eventuali approfondimenti.

Dall'interno di un sorgente Jgraph è possibile invocare un comando della shell o un programma esterno scrivendo '**`shell :`** ***comando***'; l'output generato viene contestualmente incluso nel sorgente.

Ovviamente si tratta di una caratteristica estremamente interessante e potente, poiché permette di utilizzare strumenti come Perl, Awk e simili per il tracciamento di grafici di funzione, estrazione e manipolazione di dati, ecc.; parallelamente si tratta di una caratteristica **potenzialmente pericolosa**, in quanto Jgraph non effettua nessun controllo sul comando che viene eseguito, aprendo così il varco a possibili cavalli di Troia.
Prestare dunque un'adeguata cautela!

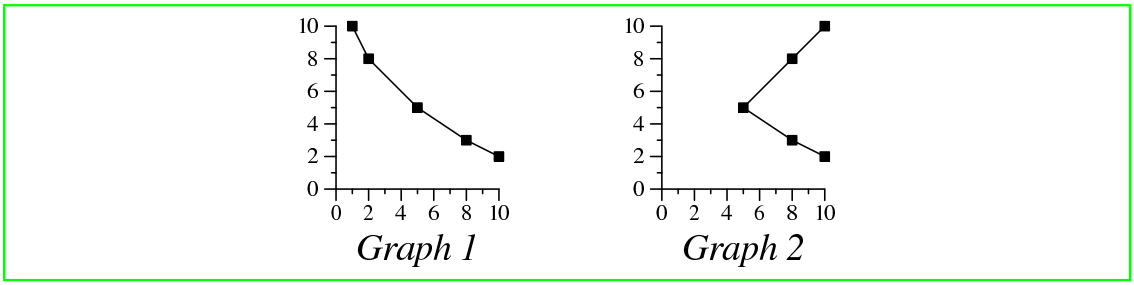
Un'altra caratteristica interessante di Jgraph è quella di poter utilizzare una qualunque immagine in formato EPS come indicatore, utilizzando l'attributo di curva '**`eps`**'.

Infine, si noti che è possibile tracciare più di un grafico nella stessa immagine mediante un ulteriore '**`newgraph`**' oppure mediante '**`copygraph`**': in quest'ultimo caso il grafico successivo eredita gli assi dal precedente. Il posizionamento relativo dei vari grafici è gestibile mediante gli attributi di grafico '**`x_translate`**' e '**`y_translate`**'. È altresì possibile posizionare i vari grafici su più pagine, mediante la parola chiave '**`newpage`**'.[6] Il listato 6.24 e la figura 6.25 presentano un esempio di utilizzo di '**`copygraph`**'.

Listato 6.24. Jgraph: grafici multipli.

```
newgraph
xaxis min 0 max 10 size 1 label fontsize 16 font Times-Italic : Graph 1
yaxis min 0 max 10 size 1

newcurve marktype box linetype solid pts 1 10 2 8 5 5 8 3 10 2

copygraph
x_translate 2
xaxis label : Graph 2
newcurve marktype box linetype solid pts 10 10 8 8 5 5 8 3 10 2
```

Figura 6.25.



6.12 Utilizzo programmatico

Il punto di forza dei programmi non interattivi è quello di poter essere facilmente utilizzati come anelli di una catena di strumenti o processi; la connessione delle varie parti può essere gestita semplicemente mediante generiche pipeline, o addirittura mediante specifici programmi «wrapper», progettati per uno scopo preciso, che costruiscano programmaticamente l'input da far elaborare.

6.12.1 «njgraph.pl»

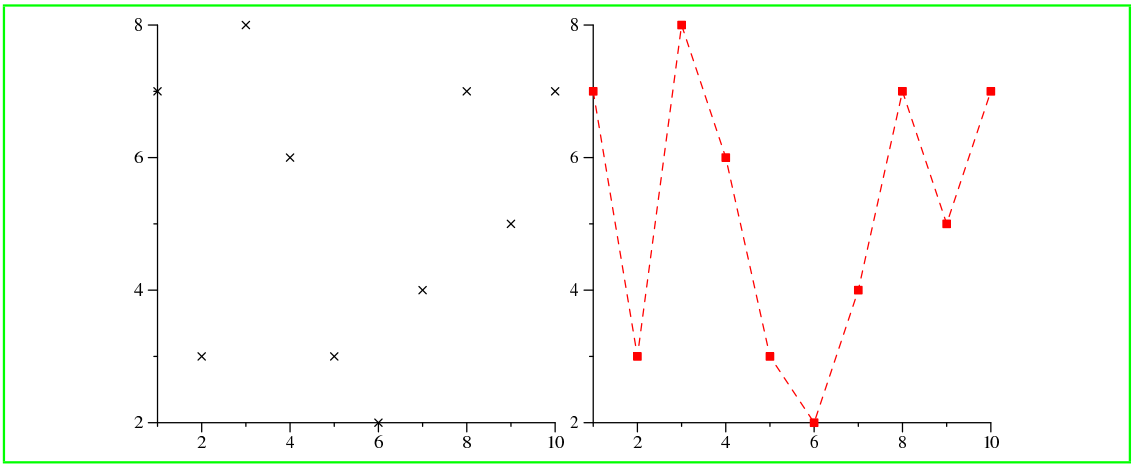
Un esempio interessante è il programma '**`njgraph.pl`**' di Neil Spring. Si tratta di un piccolo sorgente in Perl che in pratica realizza una specie di programma frontale per Jgraph: esso legge dallo standard input una singola serie di dati da rappresentare graficamente, e dopo averli opportunamente passati a Jgraph per l'elaborazione invoca GV, a meno che lo standard output non venga rediretto. Il programma '**`njgraph.pl`**' riconosce altresì gli attributi '**`marktype`**', '**`linetype`**' e '**`color`**' (e relativi valori) facoltativamente presenti sulla riga di comando, e li posiziona opportunamente all'interno del sorgente Jgraph prima dell'elaborazione.

La figura 6.26 illustra qualche esempio di invocazione del programma '**`njgraph.pl`**', mentre la figura 6.27 ne illustra l'output.

Figura 6.26. Invocazione del programma '**njgraph.pl**'.

```
$ njgraph.pl [ Invio ]

  automatically invoking gv; redirect if you want something else

1 7 2 3 3 8 4 6 5 3 6 2 7 4 8 7 9 5 10 7 [ Invio ]
[ Ctrl d ]
[ q ]
$ njgraph.pl marktype box linetype dashed color 1 0 0 [ Invio ]

  automatically invoking gv; redirect if you want something else

1 7 2 3 3 8 4 6 5 3 6 2 7 4 8 7 9 5 10 7 [ Invio ]
[ Ctrl d ]
[ q ]
$
```

Figura 6.27. Output del programma '**njgraph.pl**'.



6.12.2 «data2multijgr.sh». Qualche nota autobiografica

Pochi giorni dopo la prima pubblicazione del presente articolo, ho ricevuto un messaggio di posta elettronica da parte di un lettore. Ecco il frammento rilevante:

```
...

Per quanto riguarda la creazione dei grafici devo creare un grafico con
circa 200 barre orizzontali e pertanto devo creare un grafico che si
sviluppi su più pagine mettendo un certo numero di barre per ogni
pagina.

Distinti saluti

...
```

Ad ulteriore conferma della versatilità degli strumenti come Jgraph, ho realizzato in poco più

di mezz'ora[7] il programma '**data2multijgr.sh**' (il file 'data2multijgr_sh' contenente il programma dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro). Il programma, pur essendo molto rudimentale, risolve il problema del lettore. Eccone la sintassi:

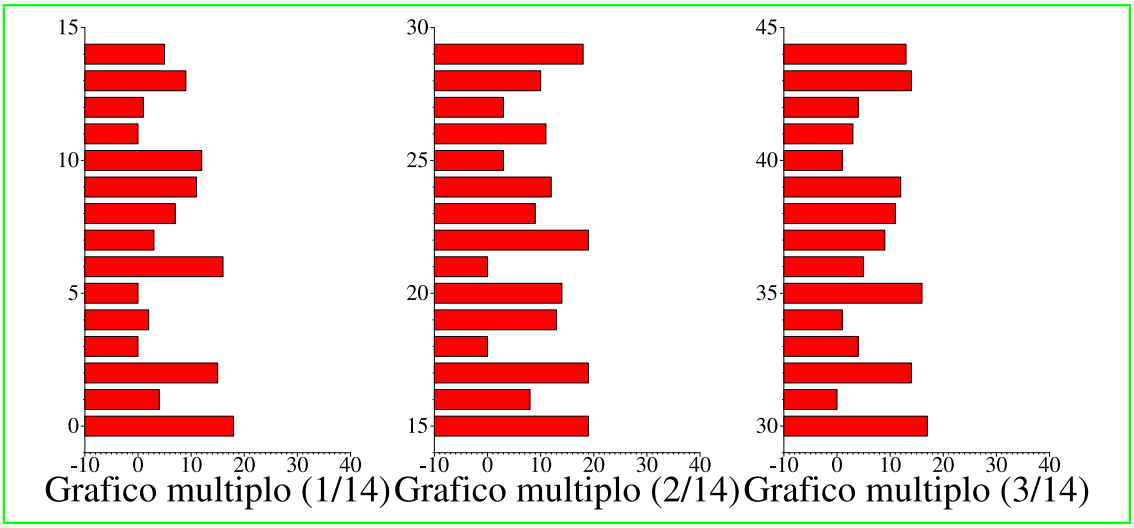
```
data2multijgr.sh [opzione] …
```

ed eccone un utilizzo tipico:

```
$ cat data [ Invio ]

  18 4 15 0 2 0 16 3 7 11 12 0 1 9 5 19 8 19 0 13 14 0 19 9 12 3 11 3 10 18
  17 0 14 4 1 16 5 9 11 12 1 3 4 14 13 1 13 13 1 5 6 7 18 17 16 10 13 19 13
  15 17 3 15 3 7 8 19 4 18 3 16 11 6 13 5 11 14 19 4 15 16 2 2 14 12 10 5 17
  9 10 12 18 5 19 13 13 8 13 17 18 8 6 9 14 19 14 6 13 5 2 1 2 17 15 8 1 18
  13 18 19 4 2 18 9 13 3 14 1 8 4 11 16 2 0 11 13 7 9 18 4 11 19 18 0 7 7 1 5
  12 11 16 8 5 6 18 19 10 4 12 18 0 4 15 2 16 6 15 15 15 6 0 18 5 10 11 4 17
  4 1 2 16 18 10 1 4 0 0 14 5 5 13 17 9 8 0 17 6 7 13 13


$ cat data | data2multijgr.sh -d 15 -t "Grafico multiplo" -X 40 -x -10 >
multi.jgr [ Invio ]

$ cat multi.jgr [ Invio ]

  newgraph
  title fontsize 50 : Grafico multiplo (1/14)
  xaxis min -10 max 40 size 5 hash_labels fontsize 30
  yaxis min -1 max 15 size 8 hash_labels fontsize 30
  newcurve marktype ybar cfill 1 0 0 marksize 1 0.75 pts
  18 0 4 1 15 2 0 3 2 4 0 5 16 6 3 7 7 8 11 9 12 10 0 11 1 12 9 13 5 14
  newpage
  newgraph
  title fontsize 50 : Grafico multiplo (2/14)
  xaxis min -10 max 40 size 5 hash_labels fontsize 30
  yaxis min 14 max 30 size 8 hash_labels fontsize 30
  newcurve marktype ybar cfill 1 0 0 marksize 1 0.75 pts
  19 15 8 16 19 17 0 18 13 19 14 20 0 21 19 22 9 23 12 24 3 25 11 26 3 27 10 28 18 29
  newpage
  newgraph
  title fontsize 50 : Grafico multiplo (3/14)
  xaxis min -10 max 40 size 5 hash_labels fontsize 30
  yaxis min 29 max 45 size 8 hash_labels fontsize 30
  newcurve marktype ybar cfill 1 0 0 marksize 1 0.75 pts
  17 30 0 31 14 32 4 33 1 34 16 35 5 36 9 37 11 38 12 39 1 40 3 41 4 42 14 43 13 44
  newpage
  newgraph

  ...


$ cat multi.jgr | jgraph -P > multi.ps [ Invio ]

$
```

Il file ‘`multi.ps`’ che così si ottiene è un file PostScript composto da più pagine ciascuna contenente un grafico relativo a un segmento specifico dei dati (la figura 6.31 riporta per brevità solamente le prime tre pagine).

Figura 6.31. Jgraph: le prime tre pagine di una grafico multiplo, distribuito su 14 pagine, realizzato grazie al programma ‘**`data2multijgr.sh`**’.



La tabella 6.32 riassume le opzioni del programma. Chiunque fosse interessato è invitato adattare il programma ‘**`data2multijgr.sh`**’ alle proprie esigenze, o aggiugervi delle caratteristiche più sofisticate (magari inviando una copia del programma modificato al sottoscritto).

Tabella 6.32. Opzioni del programma ‘**`data2multijgr.sh`**’.

Opzione	Descrizione
{`--dataperpage` | `-d`} ***dati_per_ogni_pagina***	Stabilisce quanti dati vengono inseriti nel grafico per ciascuna pagina.
{`--title` | `-t`} ***titolo***	Stabilisce il titolo da dare al grafico.
{`--xmin` | `-x`} ***valore_minimo***	Stabilisce il valore minimo sull’asse delle ascisse.
{`--xmax` | `-X`} ***valore_massimo***	Stabilisce il valore massimo sull’asse delle ascisse.
{`--ymin` | `-y`} ***valore_minimo***	Stabilisce il valore minimo sull’asse delle ordinate.
{`--ymax` | `-Y`} ***valore_massimo***	Stabilisce il valore massimo sull’asse delle ordinate.
{`--cfcolor` | `-c`} ***componente_rossa*** ↩ ↪ ***componente_verde*** ↩ ↪ ***componente_azzurra***	Imposta il colore delle barre, codificato mediante la terna RGB indicata.
{`--landscape` | `-l`}	Fa in modo che il risultato sia orientato in modo orizzontale. Il modo predefinito è quello verticale.
{`--psout` | `-p`}	Produce direttamente un file PostScript (attraverso Jgraph).
{`--view` | `-v`}	Visualizza direttamente il grafico (atraverso GV).

6.13 Riferimenti e approfondimenti

- James S. Plank, *Jgraph -- A Filter for Plotting Graphs in Postscript*
  ⟨*http://www.cs.utk.edu/~plank/plank/jgraph/jgraph.html* ⟩
- Department of Computer Science, School of Engineering, University of Virginia
  - *Jgraph*
    ⟨*http://www.cs.virginia.edu/helpnet/Authoring_Tools/jgraph.html* ⟩
  - *Jgraph manpage*
    ⟨*http://www.cs.virginia.edu/cgi-bin/manpage?section=all&topic=jgraph* ⟩
- Justin Zobel, *Example jgraph scripts*
  ⟨*http://goanna.cs.rmit.edu.au/~jz/resources/jgraph.html* ⟩
- Ajay Shah, *A collection of examples using Jgraph*
  ⟨*http://www.mayin.org/ajayshah/KB/JGRAPH/index.html* ⟩
- *Wikipedia: Bézier curve*
  ⟨*http://en.wikipedia.org/wiki/B%C3%A9zier_curve* ⟩
- Neil Spring, *Neil's Software*
  ⟨*http://www.cs.umd.edu/~nspring/software/njgraph.pl* ⟩

[1] **Jgraph** GNU GPL

[2] in pollici

[3] I colori vanno specificati in forma codificata mediante terne RGB, con valore uno che indica la massima saturazione.

[4] in unità assolute PostScript, valore predefinito 1,0

[5] con ***n*** intero positivo

[6] Ovviamente si tratta di una possibilità compatibile solo con l'output in formato PostScript (non EPS).

[7] A dire il vero il programma è stato successivamente limato, ma il prototipo funzionante era pronto dopo il tempo indicato (NdA).

Geometria euclidea con Eukleides

Eukleides[1] è un mini-linguaggio grafico per la geometria euclidea. Associati a Eukleides esistono due programmi principali:

- '**`eukleides`**', un compilatore che permette il disegno di figure geometriche all'interno di documenti TeX oppure LaTeX, utile anche per convertire tali figure in formato EPS o in altri formati grafici vettoriali;
- '**`xeukleides`**', un programma frontale per X che consente la creazione di figure geometriche interattive, utile anche per la modifica e la messa a punto del codice sorgente Eukleides.

Eukleides è stato progettato in modo da essere simile al linguaggio naturale della geometria euclidea elementare. Molto spesso è possibile evitare completamente l'utilizzo delle coordinate cartesiane. Nelle sezioni 7.4 e 7.5 vengono mostrati molti esempi che illustrano questa e altre caratteristiche del linguaggio.

Il linguaggio Eukleides e i programmi '**`eukleides`**' e '**`xeukleides`**' sono opera di Christian Obrecht, ⟨*obrecht (ad) altern·org*⟩.

Al momento della presente stesura la versione di riferimento dei programmi è la 0.9.2; Christian Obrecht sta lavorando a una ristrutturazione piuttosto completa del software, che includerà molte interessanti novità, fra cui la localizzazione del linguaggio utilizzato per le parole chiave.

Inoltre, si noti che Alml (il sistema di composizione SGML di Daniele Giacomini) offre supporto - a partire dall'estate 2005 - per il linguaggio Eukleides (vedi capitolo 4).

7.1 Guida elementare

In questa sezione vengono illustate alcune delle possibilità del programma '**eukleides**', degli script '**euk2eps**' e '**euk2edit**' e del programma '**xeukleides**'.

7.1.1 Disegno di un triangolo

Il linguaggio Eukleides è stato progettato per essere simile al linguaggio tradizionale della geometria piana euclidea. Ad esempio, per disegnare un triangolo è sufficiente preparare un file contenente il listato 7.2: se il file si chiama '`triangle.euk`', ecco come si potrebbe procedere:

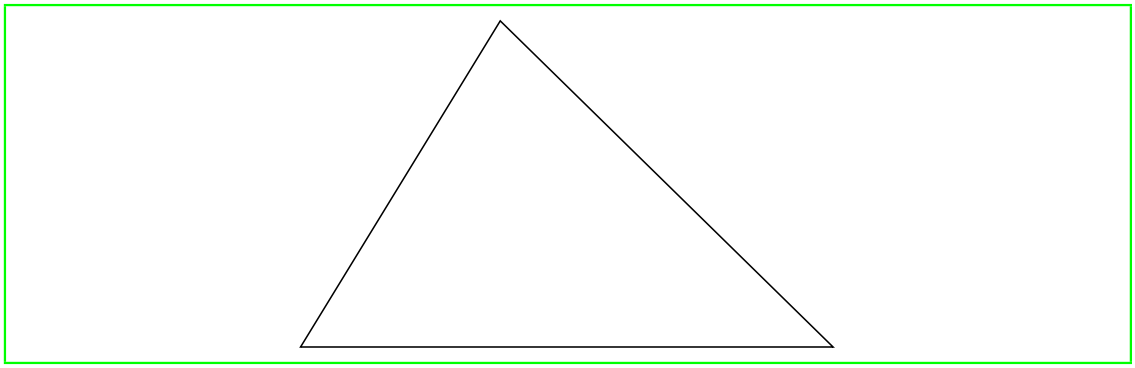
```
$ eukleides triangle.euk [ Invio ]

  % Generated by eukleides 0.9.2
  \psset{linecolor=black, linewidth=.5pt, arrowsize=2pt 4}
  \psset{unit=1.0000cm}
  \pspicture*(-2.0000,-2.0000)(8.0000,6.0000)
  \pspolygon(0.0000,0.0000)(6.0000,0.0000)(2.2500,3.6742)
  \endpspicture


$
```

L'output così generato può essere copiato così com'è in un sorgente TeX oppure LaTeX, ricordandosi però di includere anche il pacchetto '**pstricks**'. Il documento generato conterrà la figura 7.3.

Listato 7.2. Eukleides: esempio elementare.

```
A B C triangle
draw(A, B, C)
```

Figura 7.3.



Per ottenere più rapidamente una figura da visualizzare si può ricorrere allo script '**euk2eps**' il quale genera un file in formato EPS che a sua volta è visualizzabile tramite (ad esempio) GV. Esiste inoltre lo script '**euk2edit**' il quale, appoggiandosi a Pstoedit, consente la conversione in vari formati grafici vettoriali come ad esempio quello utilizzato dal programma XFig; per esempio, per poter successivamente modificare con XFig l'immagine è sufficiente procedere come segue:

$ **euk2edit triangle.euk fig** [*Invio*]

```
This is eukleides version 0.9.2
Copyright (c) Christian Obrecht 2000-2002
==> Warning: BoundingBox not found!
pstoedit: version 3.33 / DLL interface 108 (build Jul 29 2004 - release build) :
 Copyright (C) 1993 - 2003 Wolfgang Glunz
Warning: some types of raster images in the input file cannot be converted if th
e output is sent to standard output
Warning: Level 2 version of image and imagemask not supported for this backend (
due to lack of support for FILE files)
Interpreter finished. Return status 0
```

$ **cat triangle.fig** [*Invio*]

```
#FIG 3.2
Portrait
Flush left
Inches
Letter
100.00
Single
0
1200 2
# polyline
2 1 0 1 0 0 999 0 -1 4.0 0 0 0 0 0 5
        2675 3166 2143 4033 4978 4033 3206 2298 2675 3166
```

```
$
```

Spesso risulta più comodo avere un singolo file che contenga sia testo che figure; in tal caso è possibile utilizzare l'opzione '**`-f`**' e i commenti speciali '**`%--eukleides`**' e '**`%--end`**' nel sorgente TeX oppure LaTeX. Ad esempio:

```
$ cat triangle.etex [ Invio ]

  \input pstricks
  This is a scalene triangle:\par
  %--eukleides
  A B C triangle
  draw(A, B, C)
  %--end
  \bye


$ eukleides -f triangle.etex > triangle.tex [ Invio ]

$ cat triangle.tex [ Invio ]

  \input pstricks
  This is a scalene triangle:\par
  % Generated by eukleides 0.9.2
  \psset{linecolor=black, linewidth=.5pt, arrowsize=2pt 4}
  \psset{unit=1.0000cm}
  \pspicture*(-2.0000,-2.0000)(8.0000,6.0000)
  \pspolygon(0.0000,0.0000)(6.0000,0.0000)(2.2500,3.6742)
  \endpspicture
  % End of figure
  \bye


$
```

Dal file '`triangle.tex`' è quindi possibile ottenere un file in formato DVI e quindi uno in formato PostScript:

```
$ tex triangle.tex [ Invio ]

  This is TeX, Version 3.14159 (Web2C 7.4.5)
  (./triangle.tex (/usr/share/texmf/tex/generic/pstricks/pstricks.tex
  'PSTricks' v97 patch 14  <1999/12/23> (tvz)
  (/usr/share/texmf/tex/generic/pstricks/pstricks.con)) [1] )
  Output written on triangle.dvi (1 page, 740 bytes).
  Transcript written on triangle.log.


$ dvips -o triangle.ps triangle [ Invio ]

  This is dvips(k) 5.92b Copyright 2002 Radical Eye Software (www.radicaleye.com)
  ' TeX output 2006.02.05:2049' -> triangle.ps
  <texc.pro><pstricks.pro><pst-dots.pro><f7b6d320.enc><texps.pro><special.pro>
  . <cmr10.pfb>[1]
```

```
$
```

7.1.2 Ulteriori informazioni sui triangoli. Programma frontale interattivo

Il programma per X '**`xeukleides`**' è un programma frontale interattivo per il linguaggio Eukleides. Ha due modalità di funzionamento:

1. modalità di ***modifica***,
2. modalità di ***presentazione***.

Volendo avviare '**`xeukleides`**' e subito modificare il file di cui alla sezione 7.1.1, si digiti:

`$ xeukleides triangle.euk` [*Invio*]

Volendo invece avviare il programma in modalità di presentazione, è sufficiente aggiungere l'opzione '**`-V`**' alla riga di comando. Per alternare fra le due modalità si utilizi il tasto [*Esc*] (figura 7.10). Siccome la modalità di modifica è realizzata grazie alle funzionalità per la modifica del testo offerte dalle librerie GTK+, '**`xeukleides`**' possiede tutte le caratteristiche normalmente offerte da un semplice *editor* di testo; la tabella 7.11 elenca alcune utili scorciatoie.

Figura 7.10. '**`xeukleides`**': le due modalità di funzionamento (modifica e presentazione).

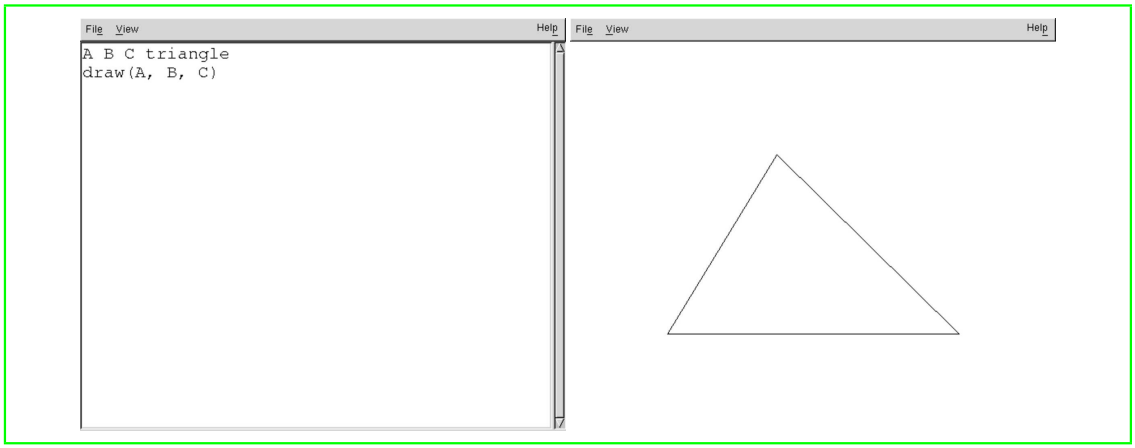


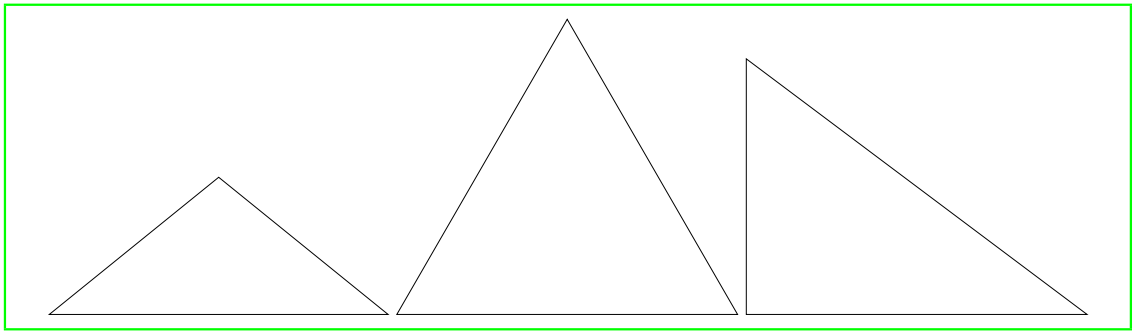
Tabella 7.11. '**`xeukleides`**': comandi utilizzabili nella modalità di modifica.

Comando	Descrizione
[*Alt F*]	Muove il cursore in avanti di una parola
[*Alt B*]	Muove il cursore indietro di una parola
[*Alt D*]	Cancella il testo fino alla fine della parola corrente
[*Ctrl W*]	Cancella il testo fino all'inizio della parola corrente
[*Ctrl U*]	Cancella il contenuto della riga corrente
[*Ctrl K*]	Cancella il testo fino alla fine della riga corrente
[*Ctrl X*]	Taglia il testo selezionato
[*Ctrl C*]	Copia il testo selezionato
[*Ctrl V*]	Incolla il testo selezionato

Il triangolo definito dal comando '**`A B C triangle`**' è un ***triangolo scaleno ottimale*** (ossia: un triangolo acutangolo la cui forma si discosta il più possibile sia da quella di un triangolo

isoscele che da quella di un triangolo rettangolo). È ovviamentre possibile disegnare altri tipi di triangolo: il linguaggio Eukleides permette di definire i triangoli in molti modi; per esempio, la figura 7.12 illustra ciò che si ottiene sostituendo il comando '**`triangle`**' con il comando '**`isosceles`**' oppure '**`equilateral`**' oppure '**`right`**'.

Figura 7.12. Eukleides: triangolo iscoscele, triangolo equilatero, triangolo rettangolo.

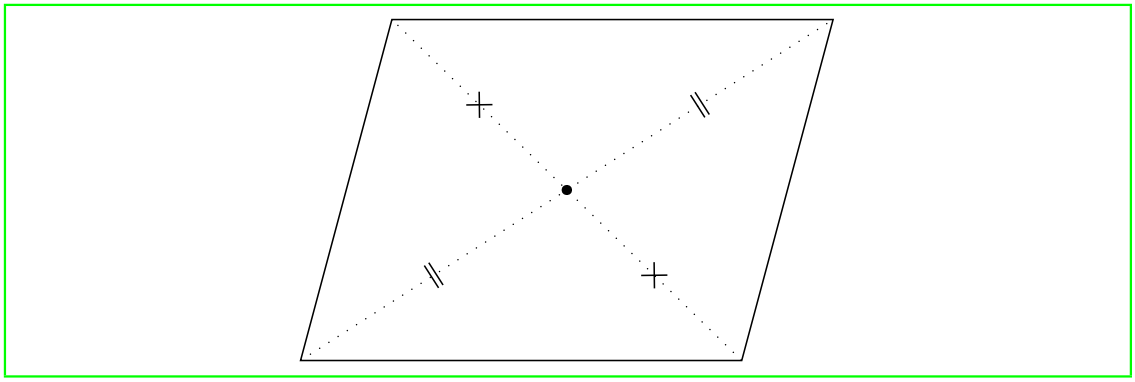


I suddetti comandi possono eventualmente accettare dei parametri opzionali, ad esempio '**`A B C triangle(6, 5, 4)`**' definisce un trinagolo ABC tale che i lati AB, BC e AC misurino rispettivamente 6 cm, 5 cm e 4 cm. Altro esempio: '**`A B C isosceles(6, 50:)`**' definisce un triangolo isoscele ABC tale che la base AB misuri 6 cm e gli angoli alla base misurino entrambi 50°.

7.1.3 Una proprietà dei parallelogrammi

In questa sezione si mostra come generare la figura 7.13.

Figura 7.13. Eukleides: le diagonali di un parallelogrammo si bisecano vicendevolmente.



Per prima cosa è necessario definire il parallelogrammo e il suo centro:

```
1  A B C D parallelogram
2  O = barycenter(A, B, C, D)
```

poi si possono disegnare entrambi:

```
3  draw(A, B, C, D) ; draw(O)
```

poi si disegnano le diagonali:

```
4  draw(segment(A, C), dotted)
5  draw(segment(B, D), dotted)
```

e infine si marcano le semidiagonali con dei doppi tratti:

```
6  mark(segment(O, A), double)
7  mark(segment(O, C), double)
```

e delle croci:

```
8  mark(segment(O, B), cross)
9  mark(segment(O, D), cross)
```

7.1.4 Una proprietà dei triangoli. Geometria interattiva

In questa sezione verrà introdotto un esempio di figura interattiva, tramite la quale si «dimostrerà» che un triangolo inscritto in una semicirconferenza è necessariamente rettangolo. Nella figura sarà possibile spostare il vertice che giace sulla semicirconferenza utilizzando i tasti [*freccia destra*] e [*freccia sinistra*].

Per prima cosa si definiscono i punti A e B e la circonferenza C di diametro AB, con i seguenti comandi:

```
1  A = point(0, 0) ; B = point(6, 0)
2  C = circle(A, B)
```

I punti A e B sono definiti tramite le loro coordinate cartesiane.[2] Successivamente si definisce una ***variabile interattiva***, ***t***:

```
3  t interactive(60, -2, 0, 180, "A", right)
```

Ciò significa che il valore iniziale di ***t*** è 60, il valore minimo è zero e il valore massimo è 180. In modalità di presentazione, ogniqualvolta si preme uno dei tasti [*freccia destra*] o [*freccia sinstra*] il valore `-2` viene addizionato o sottratto (rispettivamente) alla variabile. Indicando '**`up`**' al posto di '**`right`**' si associa la variabile ai tasti [*freccia su*] e [*freccia giù*]. Infine, '**`"A"`**' indica lo ***stato*** interno corrispondente alla variabile.[3]

Successivamente si definisce un punto M sulla circonferenza C:

```
4  M = point(C, t:)
```

Il secondo parametro è seguito dal simbolo di due punti ('`:`') per indicare che si tratta di un ***parametro angolare***: la variabile ***t*** corrisponde all'***argomento***[4] (seconda coordinata polare) del punto M rispetto al polo posto nel centro di C.

Dopodiché si disegna la semicirconferenza superiore di C e il triangolo ABM inscritto:

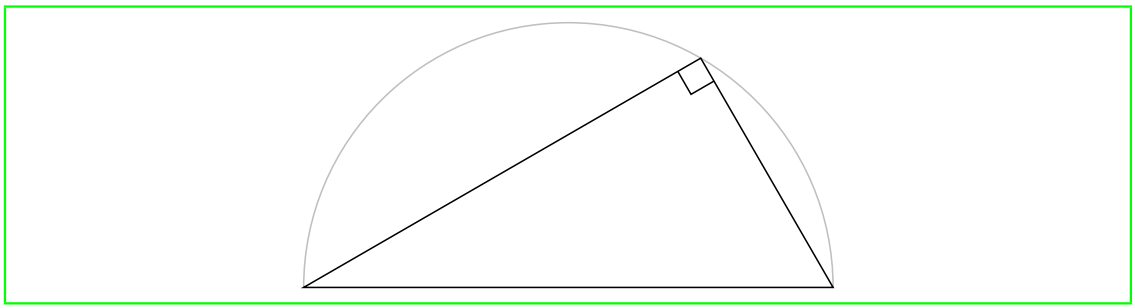
```
5  color(lightgray)
6  draw(C, 0:, 180:)
7  color(black)
8  draw(A, B, M)
```

Infine si marca l'angolo con vertice in M come retto:

```
9 | mark(A, M, B, right)
```

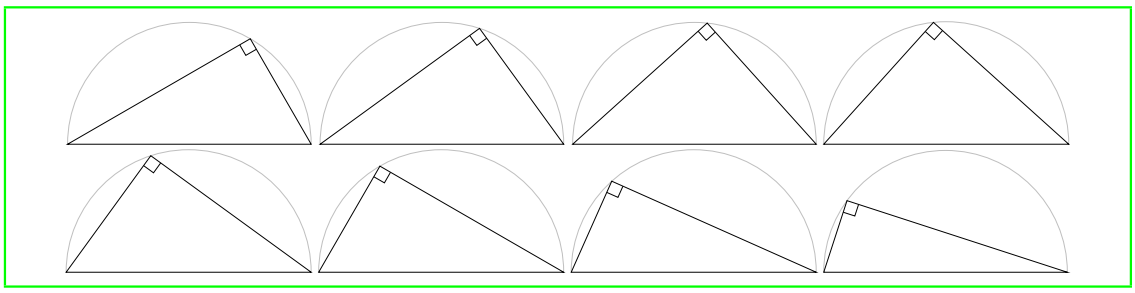
La figura 7.24 illustra il risultato finale.

Figura 7.24. Eukleides: un triangolo inscritto in una semicirconferenza è necessariamente un triangolo rettangolo.



È ora possibile interagire con la figura, grazie al programma '**xeukleides**', utilizzando i tasti freccia, e constatare infine graficamente la validità della congettura geometrica enunciata all'inizio (figura 7.25).

Figura 7.25. Eukleides: figure generate interattivamente tramite '**xeukleides**'.



Se si preme il tasto [*F1*] in modalità di presentazione si «cattura» il valore modificato della variabile interattiva: esso costituirà il nuovo valore iniziale della variabile, e il programma passa automaticamente alla modalità di modifica.

Il codice scritto per '**xeukleides**' può essere utilizzato come sorgente per '**eukleides**': le eventuali variabili interattive assumeranno staticamente il loro valore iniziale.

7.2 Guida di riferimento

In questa sezione si descrive sistematicamente il linguaggio Eukleides.

7.2.1 Sintassi

Un ***file sorgente*** Eukleides può contenere:

- ***commenti***, ossia tutti i caratteri dal simbolo '**%**' alla fine della riga;
- ***assegnamenti semplici***, ossia un nome di variabile seguito da un simbolo '**=**' e da un'espressione;
- ***assegnamenti multipli***, ossia una sequenza di nomi di variabile, separata da spazi, seguita (senza '**=**') da un comando di assegnamento multiplo (assegnamento di un triangolo, di un poligono, di intersezione, ecc.);

- ***assegnamenti interattivi***, ossia un nome di variabile seguito dalla parola chiave '`interactive`' assieme ad alcuni parametri;
- ***comandi grafici***, ossia comandi per l'impostazione dei parametri, comandi di disegno oppure comandi di tracciamento.

I nomi di variabile sono alfanumerici e sensibili alla differenza tra le lettere maiuscole e minuscole; il primo carattere **deve** essere una lettera. Le variabili possono contenere vari tipi di oggetto: numeri, vettori, rette, segmenti, circonferenze, sezioni coniche. Una singola riga del sorgente può contenere diversi comandi o assegnamenti (separati da '`;`').

In questa sezione varranno alcune notazioni convenzionali nella descrizione dei parametri dei comandi:

- x, y, z: numeri;
- a, b: parametri angolari;
- A, B, C, D, E, F, G: punti;
- u, v: vettori;
- l, l': rette;
- s: un segmento;
- c, c': circonferenze;
- ***sc***: una sezione conica;
- ***str***: una stringa;
- f, f': indicatori.

Un ***parametro angolare*** è un'espressione numerica seguita dal simbolo '`:`' (per indicare ***gradi sessagesimali***) oppure dal simbolo '`<`' (per indicare ***gradi radianti***). Un stringa è una singola riga di testo fra doppi apici.[5]

I parametri opzionali verranno indicati fra parentesi quadre.

È inoltre opportuno fare alcune precisazioni di carattere terminologico.
Nel seguito si utilizzerà la parola «ascissa» in diversi contesti: il significato probabilmente più comune è quello di «prima coordinata in un piano cartesiano», ma il significato più generale è quello di «numero che indica (univocamente a meno di casi singolari) la posizione di un punto su una curva»; in quest'ultima definizione rientra in realtà anche il primo caso (se si identifica l'ascissa del punto con l'ascissa della sua proiezione sull'asse delle ascisse, e si considera che sull'asse delle ascisse viene fissata per l'appunto convenzionalemnte un'ascissa), nonché il caso della rappresentazione parametrica di una curva piana.
Un'altra osservazione è opportuna riguardo al termine «argomento»: in un lavoro di carattere informatico si tenderebbe ad attribuire ad esso il significato di «parametro attuale passato a una procedura o funzione», e talvolta anche nella presente sezione varrà tale significato; però, parlando di geometria, esiste un'altra possibilità, ossia quella di «seconda coordinata in un piano su cui sia stato fissato un sistema di riferimento polare».
Si ritiene che il contesto dovrebbe chiarire implicitamente il significato adottato di volta in volta; in caso contrario si specificherà esplicitamente.

7.2.2 Valori numerici

Sono disponibili le usuali operazioni matematiche, mediante i simboli: '**(**', '**)**', '**+**', '**-**', '*****', '**/**', '**^**' (elevamento a potenza).

Le funzioni numeriche disponibili sono: '**abs()**', '**sqrt()**', '**exp()**', '**ln()**', '**sin()**', '**cos()**', '**tan()**', '**asin()**', '**acos()**', '**atan()**', '**deg()**' (per convertire da radianti a gradi...), '**rad()**' (... e viceversa).

È disponibile la costante '**pi**'.

La tabella 7.26 riepiloga le ulteriori funzioni che restituiscono valori numerici.

Tabella 7.26. Eukleides: funzioni che restituiscono valori numerici.

Funzione	Note esplicative
`abscissa(`***A***`)` oppure `abscissa(`***u***`)`	Ascissa (prima coordinata cartesiana) del punto o dell'estremo del vettore.
`ordinate(`***A***`)` oppure `ordinate(`***u***`)`	Ordinata (seconda coordinata cartesiana) del punto o dell'estremo del vettore.
`distance(`***A***`, `***B***`)` oppure `distance(`***A***`, `***l***`)`	Distanza fra due punti, oppure fra un punto e una retta.
`length(`***u***`)` oppure `length(`***s***`)`	Modulo di un vettore o lunghezza di un segmento.
`radius(`***c***`)`	Raggio di una circonferenza.
`major(`***sc***`)`	Il semiasse maggiore se ***sc*** è un'ellisse; il semiasse trasverso se ***sc*** è un'iperbole; il lato retto se ***sc*** è una parabola.
`minor(`***sc***`)`	Il semiasse minore se ***sc*** è un'ellisse; il semiasse non-trasverso se ***sc*** è un'iperbole; zero se ***sc*** è una parabola.
`eccentricity(`***sc***`)`	L'eccentricità di ***sc***.
`arg(`***sc***`, `***A***`)`	Ascissa del punto ***A*** nella rappresentazione parametrica interna (vedi sezione 7.2.4) della sezione conica ***sc***. Se ***A*** non giace su ***sc*** allora la funzione restituisce l'ascissa di una proiezione del punto sulla conica: la direzione della proiezione è quella dell'asse se ***sc*** è una parabola, quella del centro se ***sc*** è un'ellisse, quella dell'asse trasverso ***sc*** è un'iperbole.
`height(`***A***`, `***B***`, `***C***`)`	Distanza fra il punto ***B*** e la retta ***AC***.

Ci sono inoltre alcune funzioni (tabella 7.27) che restituiscono la misura di un angolo.[6] Gli angoli vanno intesi in ***senso generalizzato***, pertanto '**angle()**' restituisce valori compresi fra -180° (escluso) e 180° (compreso).

Tabella 7.27. Eukleides: funzioni che restituiscono misure di angoli.

Funzione	Note esplicative
`angle(`***u***`)` oppure `angle(`***l***`)` oppure `angle(`***s***`)`	Argomento dell'oggetto (rispetto alla rappresentazione interna).
`angle(`***sc***`)`	Argomento dell'asse principale della conica ***sc***.
`angle(`***u***`, `***v***`)`	Misura dell'angolo compreso fra i vettori ***u*** e ***v***.
`angle(`***A***`, `***B***`, `***C***`)`	Misura dell'angolo $\angle ABC$. Si tratta dell'angolo definito dalla rotazione antioraria attorno al punto ***B*** che porta il punto ***A*** sulla retta ***BC***, pertanto '**`angle(`*A*`, `*B*`, `*C*`) + angle(`*C*`, `*B*`, `*A*`)`**' vale 0°, tenuto conto del fatto che è necessario riportare le misure degli angoli nell'intervallo fra -180° e 180°.

7.2.3 Vettori

Il calcolo vettoriale è possibile attraverso i simboli: '**`(`**', '**`)`**', '**`+`**', '**`-`**', '**`*`**' (moltiplicazione per un numero oppure ***prodotto scalare*** di due vettori), '**`/`**' (divisione per un numero).

La tabella 7.28 riepiloga le funzioni che restituiscono un vettore.

Tabella 7.28. Eukleides: funzioni che restituiscono un vettore.

Funzione	Note esplicative
`vector(`***x***`, `***y***`)`	Vettore definito tramite coordinate cartesiane.
`vector(`***x***`, `***a***`)`	Vettore definito tramite coordinate polari.
`vector(`***A***`, `***B***`)`	Vettore definito dal segmento orientato ***AB*** (ossia ***B***-***A***).
`vector(`***l***`)`	Versore (vettore unitario) parallelo a ***l***.
`vector(`***s***`)`	Vettore definito dal segmento ***s***.
`rotation(`***u***`, `***a***`)`	Immagine del vettore ***u*** secondo la rotazione di ampiezza ***a***.

7.2.4 Punti

La tabella 7.29 riepiloga le funzioni che restituiscono dei punti.

Tabella 7.29. Eukleides: funzioni che restituiscono dei punti.

Funzione	Note esplicative
`point(`***x***`, `***y***`)`	Punto definito mediante coordinate cartesiane.
`point(`***x***`, `***a***`)`	Punto definito mediante coordinate polari.
`point(`***l***`, `***x***`)`	Punto di ascissa ***x*** lungo la retta ***l*** (in Eukleides le rette hanno una rappresentazione interna e quindi un'orientazione implicita, v. 7.2.5).
`point(`***s***`, `***x***`)`	Punto di ascissa ***x*** lungo la retta contenente il segmento ***s*** (come le rette anche i segmenti hanno una rappresentazione interna e un'orientazione implicita; l'origine delle ascisse è il primo estremo di ***s***).
`point(`***c***`, `***a***`)`	Punto su ***c*** di argomento ***a*** (secondo l'usuale rappresentazione polare di ***c***).
`point(`***sc***`, `***x***`)`	Punto su ***sc***, ove ***x*** è l'ascissa del punto sulla curva secondo la ***rappresentazione parametrica interna*** di ***sc***: per le parabole la rappresentazione si basa sulla funzione quadratica, per le ellissi sulle funzioni seno e coseno, per le iperboli sulle funzioni secante e tangente.
`barycenter(`***A***`[, `***x***`], `***B***`[, `***y***`])`	Baricentro del segmento ***AB*** con pesi degli estremi rispettivamente ***x*** e ***y*** (valori predefiniti: 1 e 1). Vale una sintassi analoga per tre oppure quattro punti.
`intersection(`***l***`, `***l′***`)`	Intersezione delle rette ***l*** e ***l′***.
`abscissa(`***l***`, `***x***`)`	Punto di ascissa (prima coordinata cartesiana) ***x*** che appartiene alla retta ***l***.
`ordinate(`***l***`, `***y***`)`	Punto di ordinata (seconda coordinata cartesiana) ***y*** che appartiene alla retta ***l***.
`midpoint(`***s***`)`	Punto medio del segmento ***s***.
`begin(`***s***`)`	Primo estremo del segmento ***s***.
`end(`***s***`)`	Secondo estremo del segmento ***s***.
`center(`***c***`)`	Centro della circonferenza ***c***.
`center(`***sc***`)`	Centro della sezione conica ***sc***. Se ***sc*** è una parabola allora restituisce il vertice.
`orthocenter(`***A***`, `***B***`, `***C***`)`	Ortocentro del triangolo ***ABC***.
`translation(`***A***`, `***u***`)`	Immagine del punto ***A*** secondo la traslazione definita dal vettore ***u***.
`reflection(`***A***`, `***l***`)`	Immagine del punto ***A*** secondo la simmetri assiale di asse ***l***.
`rotation(`***A***`, `***B***`[, `***a***`])`	Immagine del punto A secondo la rotazione di centro ***B*** e ampiezza ***a*** (valore predefinito 180°).
`homothecy(`***A***`, `***B***`, `***x***`)`	Immagine del punto A secondo l'omotetia di centro ***B*** e rapporto ***x***.
`projection(`***A***`, `***l***`[, `***l′***`])`	Proiezione del punto ***A*** sulla retta ***l*** lungo la direzione definita da ***l′*** (direzione predefinita ortogonale a ***l***).

7.2.5 Rette

Eukleides rappresenta internamente le linee rette mediante un'ascissa, un'ordinata e un angolo che ne indica l'inclinazione secondo un sistema di riferimento polare (***argomento***), sicché le rette possiedono un'orientazione implicita. La tabella 7.2.5 riepiloga tutte le funzioni che restituiscono una retta.

Tabella 7.30. Eukleides: funzioni che restituiscono una retta.

Funzione	Note esplicative
`line(`***A***`, `***B***`)`	Retta passante per ***A*** e ***B***.
`line(`***A***`, `***u***`)`	Retta passante per ***A*** e diretta secondo il vettore ***u***.
`line(`***s***`)`	Retta contenente il segmento ***s***.
`line(`***A***`, `***a***`)`	Retta passante per ***A*** e argomento ***a***.
`line(`***c***`, `***a***`)`	Retta tangente a ***c*** nel punto di argomento ***a*** (con polo coincidente con il centro di ***c***).
`line(`***sc***`, `***x***`)`	Retta tangente a ***sc*** nel punto di ascissa ***x*** (rispetto alla rappresentazione parametrica interna di ***sc***: per le parabole la rappresentazione si basa sulla funzione quadratica, per le ellissi sulle funzioni seno e coseno, per le iperboli sulle funzioni secante e tangente).
`parallel(`***l***`, `***A***`)` oppure `parallel(`***s***`, `***A***`)`	Retta passante per ***A*** e parallela a ***l*** (o a ***s***).
`perpendicular(`***l***`, `***A***`)` oppure `perpendicular(`***s***`, `***A***`)`	Retta passante per ***A*** e ortogonale a ***l*** (o a ***s***).
`bisector(`***s***`)`	Asse del segmento ***s***.
`bisector(`***A***`, `***B***`, `***C***`)`	Bisettrice dell'angolo $\angle \boldsymbol{ABC}$.
`bisector(`***l***`, `***l′***`)`	Bisettrice dell'angolo acuto formato da ***l*** e ***l′***.
`altitude(`***A***`, `***B***`, `***C***`)`	Altezza del triangolo ***ABC*** passante per ***A***.
`median(`***A***`, `***B***`, `***C***`)`	Mediana del triangolo ***ABC*** passante per ***A***.
`translation(`***l***`, `***u***`)`	Immagine di ***l*** secondo la traslazione definita dal vettore ***u***.
`reflection(`***l***`, `***l′***`)`	Immagine di ***l*** secondo la simmetria assiale di asse ***l′***.
`rotation(`***l***`, `***A***`[, `***a***`]`**)**	Immagine di ***l*** secondo la rotazione di centro ***A*** e ampiezza ***a*** (valore predefinito 180°).
`homothecy(`***l***`, `***A***`, `***x***`)`	Immagine di ***l*** secondo l'omotetia di centro ***A*** e rapporto ***x***.

7.2.6 Segmenti

Eukleides rappresenta internamente i segmenti mediante i suoi estremi, pertanto i segmenti possiedono un'orientazione implicita. La tabella 7.31 riepiloga le funzioni che restituiscono un segmento.

Tabella 7.31. Eukleides: funzioni che restituiscono un segmento.

Funzione	Note esplicative
`segment(`***A***`, `***B***`)`	Segmento di primo estremo ***A*** e secondo estremo ***B***.
`segment(`***A***`, `***u***`)`	Segmento passante per ***A*** ed equipollente a ***u***.
`segment(`***A***`, `***x***`, `***a***`)`	Segmento con primo estremo ***A***, lunghezza ***x*** e argomento ***a***.
`segment(`***c***`, `***a***`)`	Segmento con primo estremo nel centro di ***c*** e secondo estremo nel punto di ***c*** di argomento ***a*** (polo nel centro di ***c***).
`translation(`***s***`, `***u***`)`	Immagine di ***s*** secondo la traslazione definita da ***u***.
`reflection(`***s***`, `***l***`)`	Immagine di ***s*** secondo la simmetria assiale di asse ***l***.
`rotation(`***s***`, `***A***`[, `***a***`])`	Immagine di ***s*** secondo la rotazione di centro ***A*** e ampiezza ***a*** (valore predefinito 180°).
`homothecy(`***s***`, `***A***`, `***x***`)`	Immagine di ***s*** secondo l'omotetia di centro ***A*** e rapporto ***x***.

7.2.7 Circonferenze

La tabella 7.32 riepiloga le funzioni che restituiscono una circonferenza.

Tabella 7.32. Eukleides: funzioni che restituiscono una circonferenza.

Funzione	Note esplicative
`circle(`***A***`, `***B***`)`	Circonferenza di diametro ***AB***.
`circle(`***A***`, `***B***`, `***C***`)`	Circonferenza circoscritta al triangolo ***ABC***.
`circle(`***A***`, `***x***`)`	Circonferenza di centro ***A*** e raggio ***x***.
`incircle(`***A***`, `***B***`, `***C***`)`	Circonferenza inscritta nel triangolo ***ABC***.
`translation(`***c***`, `***u***`)`	Immagine di ***c*** secondo la traslazione definita da ***u***.
`reflection(`***c***`, `***l***`)`	Immagine di ***c*** secondo la simmetria assiale di asse ***l***.
`rotation(`***c***`, `***A***`[, `***a***`])`	Immagine di ***c*** secondo la rotazione di centro ***A*** e ampiezza ***a*** (valore predefinito 180°).
`homothecy(`***c***`, `***A***`, `***x***`)`	Immagine di ***c*** secondo l'omotetia di centro ***A*** e rapporto ***x***.

Ci sono inoltre i due assegnamenti multipli descritti nella tabella 7.33.

Tabella 7.33. Eukleides: assegnamenti che restituiscono le intersezioni relative a circonferenze e rette.

Assegnamento	Note esplicative
`A B intersection(l, c)`	Se i due oggetti sono tangenti allora ***A*** e ***B*** denoteranno il medesimo punto.
`A B intersection(c, c')`	Se i due oggetti sono tangenti allora ***A*** e ***B*** denoteranno il medesimo punto.

7.2.8 Sezioni coniche

La tabella 7.34 riepiloga le funzioni che restituiscono delle sezioni coniche.

Tabella 7.34. Eukleides: funzioni che restituiscono delle sezioni coniche.

Funzione	Note esplicative
`conic(A, l, x)`	Sezione conica con fuoco ***A***, direttrice ***l***, eccentricità ***x***.
`conic(A, B, x)`	Sezione conica con fuochi ***A*** e ***B***. Il valore ***x*** è la semidistanza fra i vertici (ossia l'asse maggiore per le ellissi, l'asse trasverso per le iperboli).
`parabola(A, l)`	Parabola con fuoco ***A*** e direttrice ***l***.
`parabola(A, x, a)`	Parabola con vertice ***A***, con ***x*** pari al lato retto e ***a*** pari all'argomento dell'asse.
`ellipse(A, x, y, a)`	Ellisse con centro ***A***, semiassi maggiore e minore ***x*** e ***y***, argomento dell'asse principale pari a ***a***.
`hyperbola(A, x, y, a)`	Iperbole con centro ***A***, semiassi reale e immaginario ***x*** e ***y***, argomento dell'asse principale pari a ***a***.
`translation(sc, u)`	Immagine di ***sc*** secondo la traslazione definita da ***u***.
`reflection(sc, l)`	Immagine di ***sc*** secondo la simmetria assiale di asse ***l***.
`rotation(sc, A[, a])`	Immagine di ***sc*** secondo la rotazione di centro ***A*** e ampiezza ***a*** (valore predefinito 180°).
`homothecy(sc, A, x)`	Immagine di ***sc*** secondo l'omotetia di centro ***A*** e rapporto ***x***.

La tabella 7.35 riepiloga alcuni assegamenti multipli che coinvolgono sezioni coniche.

Tabella 7.35. Eukleides: assegnamenti multipli che coinvolgono sezioni coniche.

Assegnamento	Note esplicative
`A B foci(sc)`	Fuochi della sezione conica ***sc***.
`A B vertices(sc)`	Vertici della sezione conica ***sc***.
`A B intersection(l, sc)`	Punti di intersezione fra ***l*** e ***sc***.

7.2.9 Triangoli

Un ***assegnamento di un triangolo*** è costituito da una lista di tre nomi di variabile seguita da una delle parole chiave '**`triangle`**', '**`right`**', '**`isosceles`**' oppure '**`equilateral`**' più alcuni parametri opzionali. Se alla prima variabile è già stato assegnato un punto allora il triangolo viene costruito a partire da tale punto, altrimenti si parte dall'origine. Il paramentro opzionale ***b*** è l'argomento del segmento ***AB*** (valore predefinito: 0°).

Nella tabella 7.36 vengono illustrati i vari modi per definire un triangolo.

Tabella 7.36. Eukleides: i vari modi per definire un triangolo.

Assegnamento	Note esplicative
A B C `triangle[(`*x*`[, `*b*`])]`	Definisce un triangolo scaleno tale che ***AB*** misuri ***x*** (valore predefinito: 6). Il triangolo scaleno è ***ottimale***, ossia è un triangolo acutangolo la cui forma si discosti il più possibile da quella di un triangolo rettangolo o isoscele.
A B C `triangle(`*x*`, `*y*`, `*z*`[, `*b*`])`	Definisce un triangolo scaleno tale che ***AB*** misuri ***x***, ***BC*** misuri ***y*** e ***AC*** misuri ***z***.
A B C `triangle(`*x*`, `*a*`, `*a′*`[, `*b*`])`	Definisce un triangolo scaleno tale che ***AB*** misuri ***x***, $\angle$***BAC*** misuri ***a*** e $\angle$***CBA*** misuri ***a′***.
A B C `right[(`*x*`, `*y*`[, `*b*`])]`	Definisce un triangolo rettangolo in ***A*** tale che ***AB*** misuri ***x*** e ***AC*** misuri ***y*** (valori predefiniti: 6 e 4,5 rispettivamente).
A B C `right(`*x*`, `*a*`[, `*b*`])`	Definisce un triangolo rettangolo in ***B*** tale che ***AB*** misuri ***x*** e la misura di $\angle$***BAC*** sia ***a***.
A B C `isosceles[(`*x*`, `*a*`[, `*b*`])]`	Definisce un triangolo isoscele tale che la base ***AB*** misuri ***x*** e gli angoli alla base misurino ***a*** (valori predefiniti: 6 e 39° rispettivamente).
A B C `isosceles(`*x*`, `*y*`[, `*b*`])`	Definisce un triangolo isoscele tale che la base ***AB*** misuri ***x*** e i lati obliqui misurino ***y***.
A B C `equilateral[(`*x*`[, `*b*`])]`	Definisce un triangolo equilatero di lato ***x*** (valore predefinito: 6).

7.2.10 Poligoni

Un ***assegnamento di un quadrilatero*** è costituito da una lista di quattro nomi di variabile seguita da una delle parole chiave '**`parallelogram`**', '**`rectangle`**' oppure '**`square`**' più alcuni parametri opzionali. Se alla prima variabile è di già stato assegnato un punto allora il quadrilatero viene costruito a partire da tale punto, altrimenti si parte dall'origine. Il paramentro opzionale ***b*** è l'argomento del segmento ***AB*** (valore predefinito: 0°).

Nella tabella 7.37 vengono illustrati i vari modi per definire un quadrilatero.

Tabella 7.37. Eukleides: i vari modi per definire un quadrilatero.

Assegnamento	Note esplicative
A B C D `parallelogram[(`*x*`, `*y*`, `*a*`[, `*b*`])]`	Definisce un parallelogrammo tale che ***AB*** misuri ***x***, ***AD*** misuri ***y*** e l'angolo $\angle$***BAD*** misuri ***a*** (valori predefiniti: 5, 4 e 75° rispettivamente).
A B C D `parallelogram(`*u*`, `*v*`[, `*b*`])`	Definisce un parallelogrammo tale che ***B***-***A*** = ***u*** e ***D***-***A*** = ***v***.
A B C D `rectangle[(`*x*`, `*y*`[, `*b*`])]`	Definisce un rettangolo tale che ***AB*** misuri ***x*** e ***AD*** misuri ***y*** (valori predefiniti: 6 e 6/φ rispettivamente, ove φ è il ***rapporto aureo***).

Assegnamento	Note esplicative
A B C D `square[(`*x*`[, b])]`	Definisce un quadrato di lato ***x*** (valore predefinito: 4).

Esistono inoltre l'***assegnamento di un pentagono*** o ***di un esagono*** (tabella 7.38).

Tabella 7.38. Eukleides: assegnamenti per pentagoni e per esagoni.

Assegnamento	Note esplicative
A B C D E `pentagon(`*F, x, a*`)`	Definisce un pentagono regolare di centro ***F***, raggio ***x*** e tale che l'argomento di ***FA*** sia ***a***.
A B C D E F `pentagon(`*G, x, a*`)`	Definisce un esagono regolare di centro ***G***, raggio ***x*** e tale che l'argomento di ***GA*** sia ***a***.

7.2.11 Assegnamenti interattivi

Un ***assegnamento interattivo*** è fatto così:

```
x interactive(y, z[, x', x'',] str, f)
```

Per '**eukleides**' tale assegnamento equivale a '*x* **=** *y*'; usando '**xeukleides**' esso permette di modificare il valore di ***x*** **dalla modalità di presentazione** mediante i tasti freccia. Il valore iniziale di ***x*** è ***y*** e l'incremento è ***z***. I parametri opzionali ***x′*** e ***x″*** stabiliscono il minimo e il massimo per ***x***. La stringa ***str*** **deve** consistere di un singolo carattere alfabetico maiuscolo: per modificare ***x*** si deve premere per prima cosa il tasto alfabetico corrispondente.[7] I valori possibili per ***f*** sono '**right**' (nel qual caso ***x*** viene incrementato alla pressione del tasto [*freccia destra*] e decrementato alla pressione del tasto [*freccia sinistra*]) oppure '**up**' (nel qual caso ***x*** viene incrementato alla pressione del tasto [*freccia su*] e decrementato alla pressione del tasto [*freccia giù*]).

7.2.12 Comandi di impostazione

La tabella 7.39 riepiloga comandi di impostazione.

Tabella 7.39. Eukleides: comandi di impostazione.

Comando	Note esplicative
`{box | frame}(`*x, y, x′, y′*`[, z])`	Con '**eukleides**' i due comandi hanno un effetto simile, ossia impostare il riquadro visibile. Il vertice inferiore sinistro ha coordinate cartesiane (***x***,***y***) e quello superiore destro coordinate (***x′***,***y′***) mentre il parametro opzionale imposta l'unità di misura a ***z*** cm (valore predefinito: 1). Con '**xeukleides**' la figura viene disegnata utilizzando la scala massima che renda il riquadro inscritto nell'area di disegno. Il comando '**box**' imposta il riquadro visibile sicché la figura di solito appare con due strisce grigie ai lati. Il comando '**frame**' imposta il **minimo** riquadro visibile sicché viene utilizzata l'intera area di disegno. Il parametro ***z*** viene ignorato. Il riquadro predefinito è: (-2, -2, 8, 6).
`color(`*f*`)`	Imposta il colore a ***f***. I valori ammessi sono '**black**' (predefinito), '**darkgray**', '**gray**', '**lightgray**', '**white**', '**red**', '**green**', '**blue**', '**cyan**', '**magenta**' e '**yellow**'.

Comando	Note esplicative
`color` ***str***	Con '**eukleides**' imposta il colore a ***str*** (dev'essere un colore valido per '**pstricks**'). Con '**xeukleides**' non ha effetto.
`thickness(`***x***`)`	Con '**eukleides**' imposta lo spessore delle linee corrente a ***x*** pt (lo spessore predefinito è 0,5 pt). Con '**xeukleides**' non ha effetto.
`style(`***f***`)`	Imposta lo stile di disegno corrente a ***f***. I valori ammessi sono '**full**', '**dashed**' e '**dotted**'. Il valore predefinito è '**full**'.
`tricks` ***str***	Con '**eukleides**' aggiunge in testa a ***str*** un carattere '\' e emette il risultato. Deve trattarsi di codice TeX valido. Con '**xeukleides**' non ha effetto.
`font` ***str***	Con '**xeukleides**' cambia la fonte corrente. La stringa ***str*** deve essere un valido nome XLFD (*X Logical Font Description Conventions*). Con '**eukleides**' non ha effetto.

7.2.13 Comandi di disegno

La tabella 7.40 riepiloga comandi di disegno.

Tabella 7.40. Eukleides: comandi di disegno.

Comando	Note esplicative
`draw(`***A***`[, `***f***`[, `***x***`]])`	Disegna un punto ***A*** con forma definita da ***f*** e rapporto di scala definito da ***x*** (valore predefinito: 1). I valori ammessi per ***f*** sono '**dot**', '**box**', '**cross**' e '**plus**' (valore predefinito: '**dot**'). Il rapporto di scala non ha effetto con '**xeukleides**'.
`draw(`***u***`, `***A***`[, `***f***`])`	Disegna un vettore ***u*** con origine in ***A***. Specificando ***f*** si può modificare lo stile corrente; i valori ammessi sono: '**full**', '**dashed**' e '**dotted**'.
`draw(`***l***`[, `***f***`[, `***f′***`]])`	Disegna una retta. Specificando ***f*** o ***f′*** si può modificare lo stile corrente: i valori ammessi per ***f*** sono '**full**', '**dashed**' e '**dotted**'; i valori ammessi per ***f′*** sono '**entire**', '**halfline**' e '**backhalfline**' (valore predefinito: '**entire**').
`draw(`***s***`[, `***f***`[, `***f′***`]])`	Disegna un segmento. Specificando ***f*** o ***f′*** si può modificare lo stile corrente: i valori ammessi per ***f*** sono '**full**', '**dashed**' e '**dotted**'; i valori ammessi per ***f′*** sono '**noarrow**', '**arrow**', '**backarrow**' e '**doublearrow**' (valore predefinito: '**noarrow**').
`draw(`***c***`[, `***f***`])`	Disegna una circonferenza. Specificando ***f*** si può modificare lo stile corrente: i valori ammessi sono '**full**', '**dashed**' e '**dotted**'.
`draw(`***c***`, `***a***`, `***a′***`[, `***f***`[, `***f′***`]])`	Disegna arco della circonferenza ***c*** compreso fra i punti di coordinate polari angolari ***a*** e ***a′***. Specificando ***f*** o ***f′*** si può modificare lo stile corrente: i valori ammessi per ***f*** sono '**full**', '**dashed**' e '**dotted**'; i valori ammessi per ***f′*** sono '**noarrow**', '**arrow**', '**backarrow**' e '**doublearrow**' (valore predefinito: '**noarrow**').
`draw(`***sc***`[, `***f***`])`	Disegna una sezione conica ***sc***. Specificando ***f*** si può modificare lo stile corrente: i valori ammessi sono '**full**', '**dashed**' e '**dotted**'.
`draw(`***sc***`, `***x***`, `***y***`[, `***f***`])`	Disegna un arco della sezione conica ***sc***. I valori ***x*** e ***y*** denotano l'ascissa (relativamente alla rappresentazione parametrica interna di ***sc***) degli estremi dell'arco. Specificando ***f*** si può modificare lo stile corrente: i valori ammessi sono '**full**', '**dashed**' e '**dotted**'.

Comando	Note esplicative
`draw(`***A***`, `***B***`, `***C***`[, `***f***`])`	Disegna il triangolo ***ABC***. Specificando ***f*** si può modificare lo stile corrente: i valori ammessi sono '**`full`**', '**`dashed`**' e '**`dotted`**'.
`draw(`***A***`, `***B***`, `***C***`, `***D***`[, `***f***`])`	Disegna il quadrilatero ***ABCD***. Specificando ***f*** si può modificare lo stile corrente: i valori ammessi sono '**`full`**', '**`dashed`**' e '**`dotted`**'.
`draw(`***A***`, `***B***`, `***C***`, `***D***`, `***E***`[, `***f***`])`	Disegna il pentagono ***ABCDE***. Specificando ***f*** si può modificare lo stile corrente: i valori ammessi sono '**`full`**', '**`dashed`**' e '**`dotted`**'.
`draw(`***A***`, `***B***`, `***C***`, `***D***`, `***E***`, `***F***`[, `***f***`])`	Disegna l'esagono ***ABCDEF***. Specificando ***f*** si può modificare lo stile corrente: i valori ammessi sono '**`full`**', '**`dashed`**' e '**`dotted`**'.
`draw(`***str***`, `***A***`,[ `***x***`,] `***a***`)`	Scrive il testo contenuto in ***str*** a una distanza ***x*** (valore predefinito: 0,3 cm) e con argomento ***a*** rispetto al polo ***A***.
`draw(`***str***`, `***s***`,[ `***x***`,] `***a***`)`	Scrive il testo contenuto in ***str*** a una distanza ***x*** (valore predefinito: 0,3 cm) e con argomento ***a*** rispetto al punto medio del segmento ***s***.
`draw(`***x***`,[ `***str***`,] `***A***`,[ `***y***`,] `***a***`)`	Scrive il valore di ***x***, eventualmente secondo il formato definito da ***str*** (sintassi del linguaggio C), a una distanza ***y*** (valore predefinito 0,3 cm) e con argomento ***a*** rispetto al polo ***A***.
`draw(`***x***`,[ `***str***`,] `***s***`,[ `***y***`,] `***a***`)`	Scrive il valore di ***x***, eventualmente secondo il formato definito da ***str*** (sintassi del linguaggio C), a una distanza ***y*** (valore predefinito 0,3 cm) e con argomento ***a*** rispetto al punto medio del segmento ***s***.
`draw(`***x***`, `x'`, `***str***`, `***A***`,[ `***y***`,] `***a***`)`	Scrive i valori di ***x*** e x', eventualmente secondo il formato definito da ***str*** (sintassi del linguaggio C), a una distanza ***y*** (valore predefinito 0,3 cm) e con argomento ***a*** rispetto al polo ***A***.
`draw(`***x***`, `x'`, `***str***`, `***s***`,[ `***y***`,] `***a***`)`	Scrive i valori di ***x*** e x', eventualmente secondo il formato definito da ***str*** (sintassi del linguaggio C), a una distanza ***y*** (valore predefinito 0,3 cm) e con argomento ***a*** rispetto al punto medio del segmento ***s***.
`mark(`***s***`[, `***f***`[, `***x***`]])`	Marca il segmento ***s*** con un simbolo definito da ***f*** usando un rapporto di scala ***x*** (valore predefinito: 1). I valori ammessi per ***f*** sono: '**`simple`**', '**`double`**', '**`triple`**' e '**`cross`**'.
`mark(`***A***`, `***B***`, `***C***`[, `***f***`[, `***x***`]])`	Marca l'angolo $\angle$***ABC*** con un simbolo definito da ***f*** usando un rapporto di scala ***x*** (valore predefinito: 1). I valori ammessi per ***f*** sono: '**`simple`**', '**`double`**', '**`triple`**', '**`dashed`**' (quest'ultimo produce una marcatore curvo con un tratto trasversale), '**`right`**', '**`dotted`**' (quest'ultimo produce una marcatore di angolo retto con un punto al centro), '**`arrow`**' e '**`backarrow`**' (valore predefinito: '**`simple`**').

7.2.14 Comandi di tracciamento

Un ***comando di tracciamento*** permette di disegnare un luogo geometrico al variare di un parametro. Ecco la sintassi:

```
trace(x, x', x''[, f])
```

Il comando deve essere seguito da almeno un'espressione che restituisca un punto, inclusa fra parentesi graffe; l'espressione deve dipendere dalla variabile ***x***. I paramteri x' e x'' denotano

rispettivamente il minimo e il massimo per ***x***. L'espressione deve essere definita in tutto l'intervallo, altrimenti si possono ottenere risultati incoerenti o a volte (con '**eukleides**') dei messaggi di errore. All'interno delle parentesi graffe, al fine di eseguire calcoli intermedi, è possibile far precedere qualsiasi successione di comandi all'espressione. Specificando ***f*** si può modificare lo stile corrente: i valori ammessi sono '**full**', '**dashed**' e '**dotted**'.

7.2.15 «pstricks»

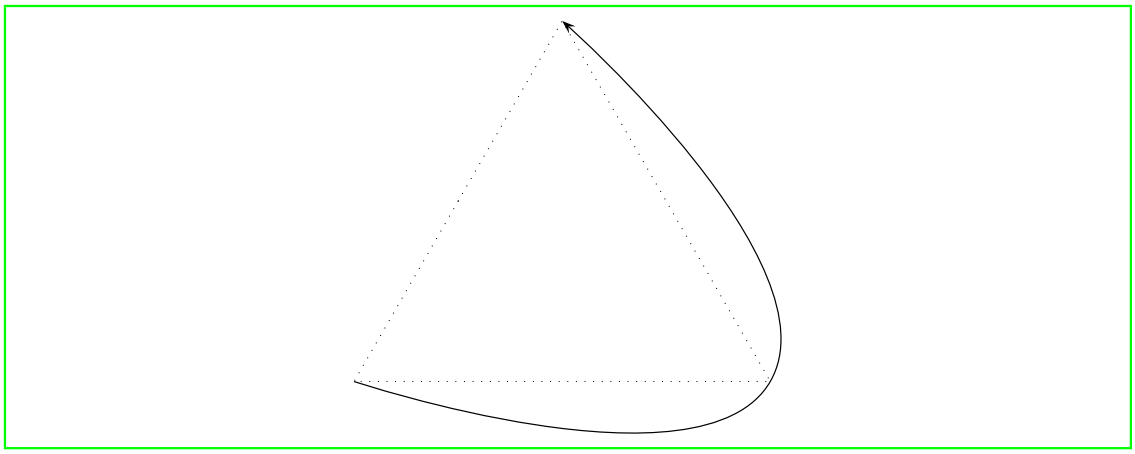
Il programma '**eukleides**' permette di includere dei comandi del pacchetto di macro '**pstricks**' utilizzando parametri gestiti da '**eukleides**' stesso. L'utilizzo probabilmente più opportuno di questa funzionalità è quello di disegnare degli oggetti '**pstricks**' con coordinate generate da una costruzione in linguaggio Eukleides.

La sintassi è ragionevolmente simile a quella che si utilizza nell'inserire codice '**pstricks**' in un documento TeX o LaTeX. L'invocazione di una macro di '**pstricks**' inizia con il simbolo '\' e prosegue con il nome della macro stessa (ad esempio: '**pscircle**'). Può esserci anche una lista di opzioni per la macro, inclusa fra parentesi quadre, e anche una lista di parametri fra parentesi tonde. Un semplice esempio è presentato nel listato 7.41 e nella corrispondente figura 7.42; nel seguito varranno date spiegazioni più dettagliate.

Listato 7.41. Eukleides: esempio elementare con '**pstricks**'.

```
A B C equilateral
draw(A, B, C, dotted)
\pscurve[arrows="->"](A, B, C)
```

Figura 7.42.



7.2.15.1 Opzioni

Le eventuali opzioni devono essere specificate nella forma

opzione=valore

dove ***opzione*** è un'opzione prevista da '**pstricks**' e ***valore*** può essere una stringa fra doppi apici (ad esempio: '**arrows="->"**') oppure un espressione numerica di Eukleides (ad esempio: '**radius=distance(A,B)**'). I doppi apici verranno eliminati dal programma prima dell'emissione del codice '**pstricks**'.

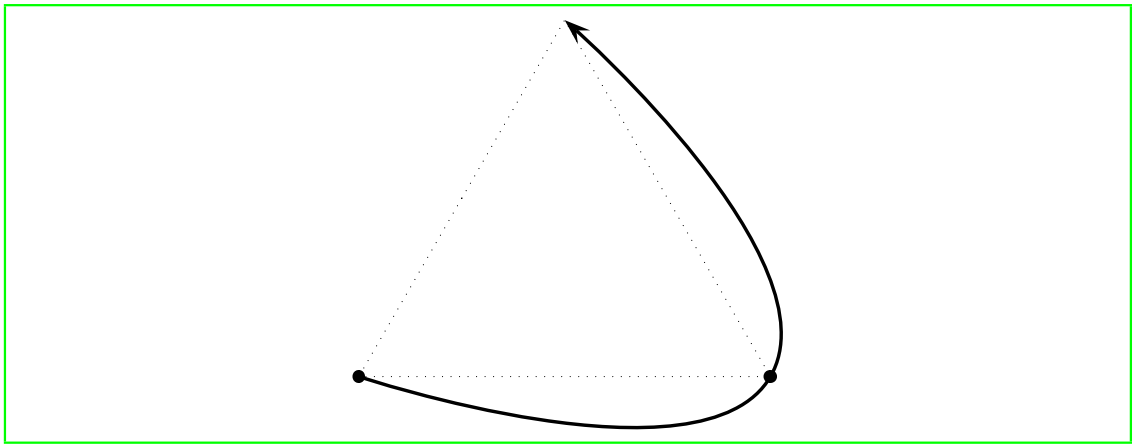
Tutte le opzioni di '**pstricks**' che siano documentate dovrebbero funzionare senza il bisogno di includerle fra doppi apici, ma in caso di problemi è possibile farlo (ad esempio: '**"parametroStrano"=3.145**').

A titolo di esempio, si considerino il listato 7.43 e la corrispondente figura 7.44 (che modificano lievemente il listato 7.41 e la figura 7.42, rispettivamente).

Listato 7.43. Eukleides: altro esempio elementare con '**pstricks**'.

```
A B C equilateral
draw(A, B, C, dotted)
\pscurve[showpoints="true", linewidth=.05, arrows="->"](A, B, C)
```

Figura 7.44.



7.2.15.2 Parametri

I parametri da passare alle macro di '**pstricks**' devono essere specificati come una lista, separata da virgole, di espressioni che restituiscano punti oppure valori numerici o di stringhe fra doppi apici. I punti vengono trasformati nella forma '**(*x*,*y*)**' e le stringhe fra doppi apici e le espressioni numeriche vengono inserite all'interno delle parentesi graffe; ad esempio, il comando '**\psline("<<->>", A, B)**' viene tradotto nel codice '**pstricks**' seguente:

```
\psline{<<->>}(0.0000,0.0000)(6.0000,0.0000)
```

Le indicazioni qui fornite non pretendono minimamente di essere una spiegazione - nemmeno superficiale - del funzionamento del pacchetto di macro '**pstricks**'. Per maggiori informazioni si veda la sezione 7.6.
Inoltre, si tenga presente che non viene effettuato nessun controllo sul codice '**pstricks**' che viene generato da '**eukleides**', il che significa che è possibile generare codice TeX sintatticamente scorretto se si utilizza questa caratteristica di '**eukleides**'.
Infine si noti che al momento della presente stesura '**xeukleides**' ignora completamente le macro '**pstricks**'.

7.3 Limitazioni e punti di forza

Le limitazioni principali di Eukleides sono probabilmente:

1. il basso livello di interattività del programma frontale '**`xeukleides`**', e
2. la mancanza di strutture sintattiche più evolute (cicli, sottoprogrammi, ecc...) nel linguaggio vero e proprio.

Per quanto riguarda il primo punto, la questione è complessa e si potrebbe inquadrare più nell'ambito delle scelte metodologiche che in quello dei compromessi tecnologici. In effetti esistono già diversi programmi che permettono di «fare geometria dinamica», nel senso che permettono all'utente di costruire e modificare gli oggetti geometrici essenzialmente «a colpi di mouse»: non è questa la sede per aprire una discussione sulla presunta opportunità didattica di strumenti di questo tipo (basti notare che al riguardo le diverse opinioni dei docenti di matematica sono largamente divergenti); fatto sta che la scelta dell'autore di Eukleides (egli stesso insegnante liceale di matematica in Francia) è stata diversa:

> As a mathematics teacher in a French high school, I have to compose a rather large number of documents for my students, containing both text and formulas. In my point of view, LaTeX is the best tool in such a situation, combining efficiency and high quality. Very often, these documents should be illustrated with geometric figures. I first used the excellent '**`pstrick`**' package to draw them. I didn't want to use WYSIWYG software instead, because I wanted to keep following TeX's philosophy, that is: What You Mean Is What You Get. Unfortunately, '**`pstrick`**' isn't designed for geometry at all and is rather inappropriate in many situations.
>
> One night, I wanted to draw a triangle with an inscribed circle, so I had to compute by hand the coordinates of the center and the radius of this circle, which is quite boring. During these calculations, I realized that they could easily be done by a computer, and that gave me the idea to create Eukleides, a geometry drawing language. *My goal was to make it as close as possible to what mathematics teachers would say to describe geometric figures.*[8]
>
> [...]

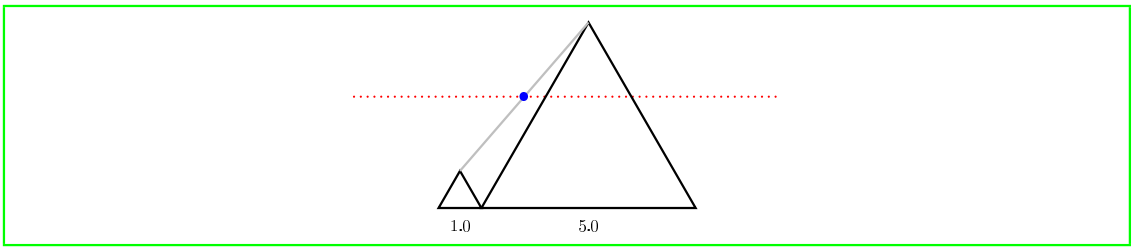
Sembra quindi che lo scopo principale di Eukleides sia quello di (facilitare la e) invitare alla **descrizione** verbale delle figure geometriche, piuttosto che coltivare un'attività semi-ludica di modifica interattiva delle figure stesse (cosa comunque praticamente possibile, anche se «senza mouse», grazie agli assegnamenti interattivi, v. sezione 7.2.11). Peraltro, avere a che fare con un software *orientato alla descrizione* offre anche tutta una serie di indiscutibili vantaggi (come si cercherà di dimostrare nel seguito).

Riguardo al secondo punto (fermo restando che le future versioni di Eukleides conterranno dei decisi miglioramenti in questo senso), proprio grazie alla natura descrittivo-testuale del software in oggetto è possibile «estendere» le funzionalità del linguaggio utilizzando la flessibilità offerta dai diversi linguaggi di programmazione. Si consideri ad esempio il seguente enunciato: «Sia dato un segmento e un punto su di esso; si costruiscano - sulle due parti del segmento così definite - due triangoli equilateri dalla stessa parte del piano rispetto al segmento, e si consideri il segmento avente per estremi i vertici non giacenti sul segmento dato; il punto medio di tale segmento descrive - al variare del punto dato - un segmento su una retta parallela al segmento dato». Per tentare di fornire una «dimostrazione dinamica» dell'enunciato, si parta dal caso particolare fornito dal listato 7.46 e dalla figura 7.47.

Listato 7.46.

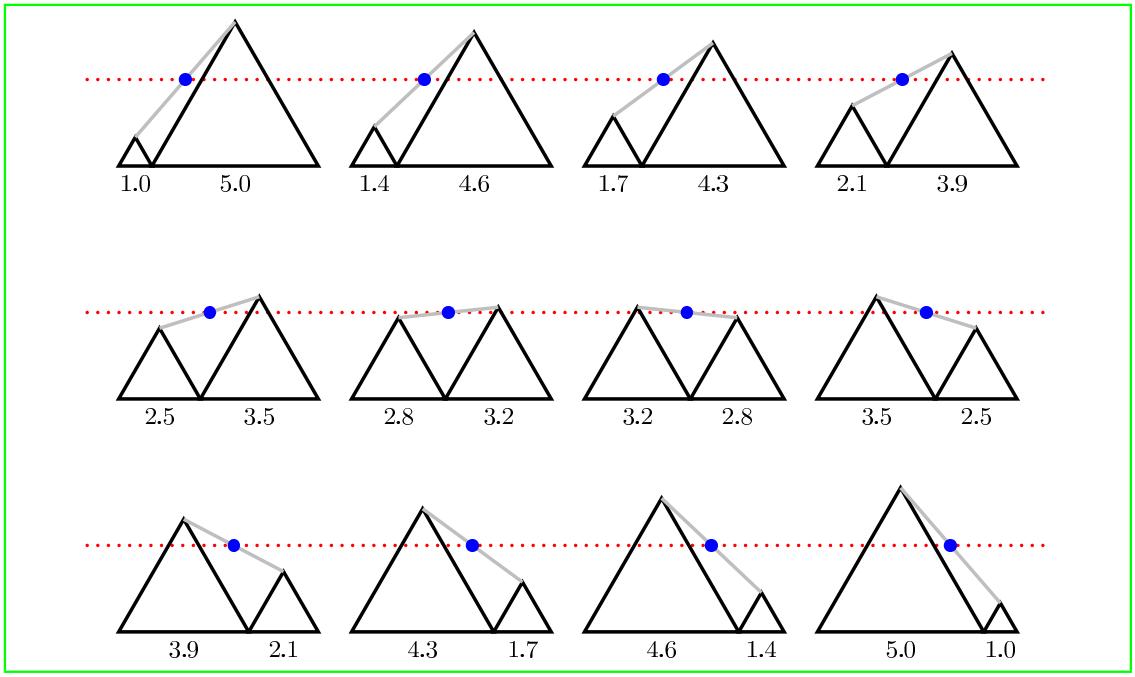
```
thickness(3)
H = point(0,3*3^(1/2)/2)
color(red)
style(dotted)
% la retta che dovrebbe contenere il luogo:
draw(line(H,0:))
color(black)
style(full)
A = point(0,0)
AM = 1
MB = 6-AM
A M I equilateral(AM)
M B J equilateral(MB)
draw(A,M,I) ; draw(M,B,J)
color(lightgray)
draw(segment(I,J))
color(black)
draw(AM,"%.1f",segment(A,M),-90:)
draw(MB,"%.1f",segment(M,B),-90:)
color(blue)
% il punto che descrive il luogo:
draw(barycenter(I,J))
```

Figura 7.47. Eukleides: un problema di luoghi geometrici (un caso particolare).



Naturalmente, una «dimostrazione dinamica» richiede di poter disegnare un gran numero di casi particolari simili a quello illustrato, mettendo così in evidenza la limitazione di Eukleides legata al non poter definire cicli enumerativi. Tuttavia, utilizzando dei linguaggi di programmazione esterni, è abbastanza semplice arrivare a figure come la 7.48.

Figura 7.48. Eukleides: una «dimostrazione dinamica».



Per realizzare tale figura, si è utilizzato il programma '**eukloop.sh**' (il file '`eukloop_sh`' contenente il programma dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro), a partire dai file '`2triloc.eukt`' e '`2triseq.eukt`'. Ecco come si procede:

`$ echo "frame(-1,-16,28,6,0.5)" > 2tri.euk ; echo "thickness(3)" >> 2tri.euk` [*Invio*]

`$ eukloop.sh` [*Invio*]

```
create symbolic link 'eukloop1.sh' to 'eukloop.sh'
create symbolic link 'eukloop2.sh' to 'eukloop.sh'
```

`$ cat 2triloc.eukt` [*Invio*]

```
y = <COUNTER1>
H = point(0,-7*y+3*3^(1/2)/2)
color(red)
style(dotted)
draw(line(H,0:))
color(black)
style(full)
```

`$ cat 2triloc.eukt | eukloop1.sh 0 1 2 >> 2tri.euk` [*Invio*]

`$ cat 2triseq.eukt` [*Invio*]

```
x = <COUNTER2>
y = <COUNTER1>
A = point(7*x,-7*y)
AM = 4/11*x+16/11*y+1
MB = 6-AM
```

```
A M I equilateral(AM)
M B J equilateral(MB)
draw(A,M,I) ; draw(M,B,J)
color(lightgray)
draw(segment(I,J))
color(black)
draw(AM,"%.1f",segment(A,M),-90:)
draw(MB,"%.1f",segment(M,B),-90:)
color(blue)
draw(barycenter(I,J))
color(black)
```

```
$ cat 2triseq.eukt | eukloop2.sh 0 1 2 0 1 3 >> 2tri.euk [ Invio ]

$
```

Il file '`2tri.euk`' dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro): esso corrisponde esattamente alla figura 7.48.

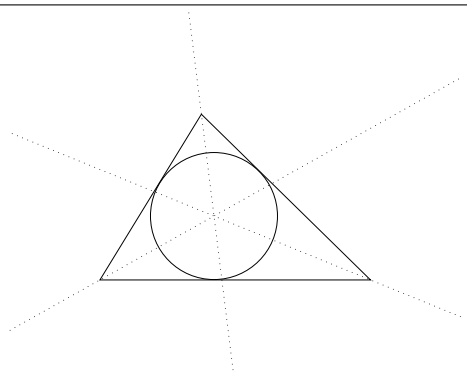
Il funzionamento del programma '**`eukloop.sh`**' è molto semplice:[9] invocandolo con il suo nome esso crea due collegamenti simbolici a se stesso nella directory corrente, che verranno poi utilizzati nell'invocazione effettiva, ciascuno di essi corrispondendo a una differente modalità di utilizzo; il primo collegamento simbolico realizza una struttura ciclica enumerativa semplice, il secondo una composta da due cicli annidati; in pratica il programma converte un file in cui sia presente del codice Eukleides assieme alle metavariabili <***COUNTER1***> e <***COUNTER2***> in un file Eukleides ottenuto ripetendo il codice Eukleides e sostituendo alle metavariabili dei valori specifici ottenuti ciclicamente in conformità agli argomenti passati al programma; nel caso di '**`eukloop1.sh`**' i tre argomenti passati corrispondono rispettivamente al valore iniziale, all'incremento e al valore finale (e analogamente nel caso di '**`eukloop2.sh`**', solo che in questo caso bisogna passare tre valori per ciascuna metavariabile, per un totale di sei argomenti); nel caso del ciclo doppio, il ciclo esterno corrisponde alla metavariabile <***COUNTER1***>, quello interno alla metavariabile <***COUNTER2***>.

7.4 Esempi

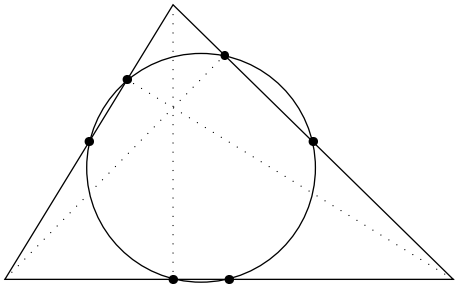
Gli esempi presentati nella tabella 7.52 illustrano le caratteristiche fondamentali di Eukleides; nella sezione 7.5 vengono presentati degli esempi più avanzati.

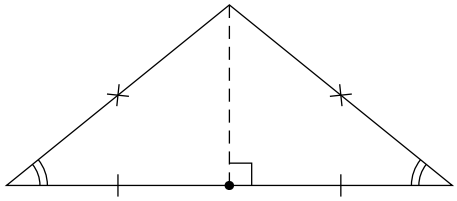
Tabella 7.52. Eukleides: esempi elementari.

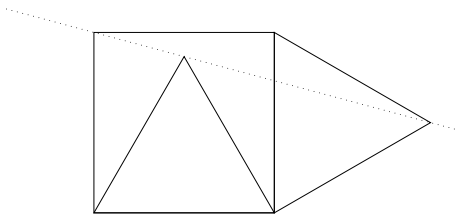
Codice Eukleides	Figura risultante

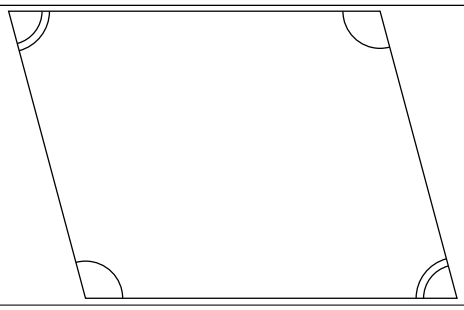
`% Circonferenza inscritta e bisettrici`
`A B C triangle`
`draw(A, B, C)`
`draw(incircle(A, B, C))`
`draw(bisector(B, A, C), dotted)`
`draw(bisector(A, B, C), dotted)`
`draw(bisector(B, C, A), dotted)`
	

Codice Eukleides	Figura risultante

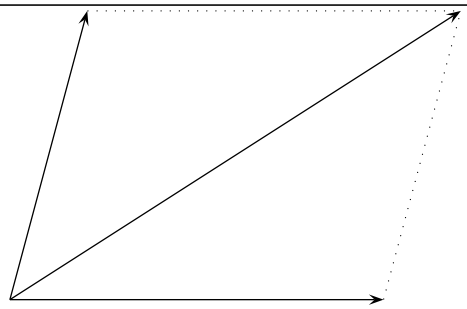
`% Circolo di Feuerbach`
`A B C triangle`
`a = projection(A, line(B, C))`
`b = projection(B, line(A, C))`
`c = projection(C, line(A, B))`
`draw(A, B, C)`
`draw(a) ; draw(b) ; draw(c)`
`draw(segment(A, a), dotted)`
`draw(segment(B, b), dotted)`
`draw(segment(C, c), dotted)`
`draw(barycenter(A, B))`
`draw(barycenter(B, C))`
`draw(barycenter(C, A))`
`draw(circle(a, b, c))`
	

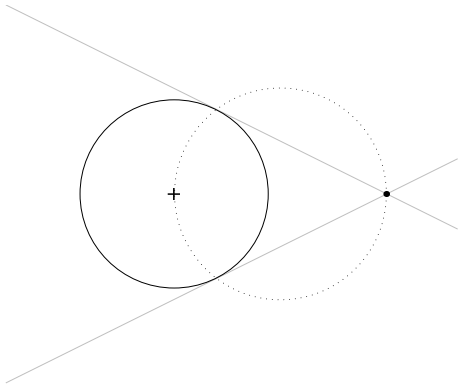
`% Triangolo isoscele`
`A B C isosceles`
`H = projection(C, line(A, B))`
`draw(A, B, C)`
`draw(H)`
`draw(segment(C, H), dashed)`
`mark(B, H, C, right)`
`mark(segment(A, H))`
`mark(segment(B, H))`
`mark(segment(A, C), cross)`
`mark(segment(C, B), cross)`
`mark(B, A, C, double)`
`mark(C, B, A, double)`
	

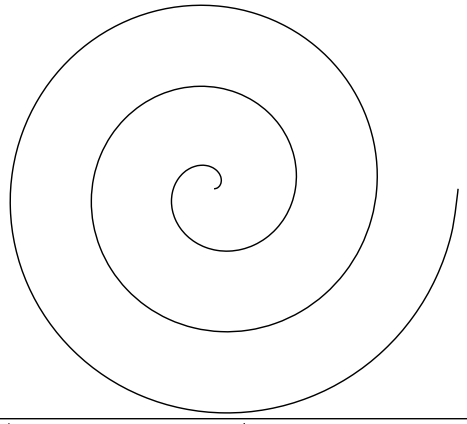
`% Punti allineati`
`A B C D square`
`A B E equilateral(4)`
`B F G equilateral(4, 30:)`
`draw(A, B, C, D)`
`draw(A, B, E)`
`draw(B, F, G)`
`draw(line(E, F), dotted)`
	

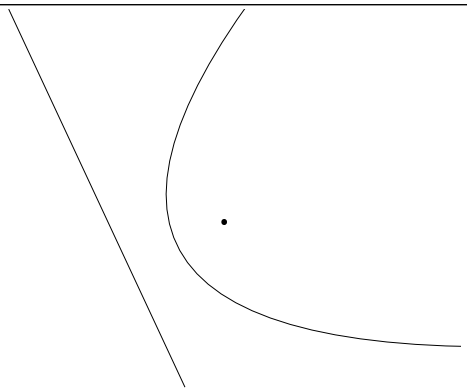
`% Angoli di un parallelogrammo`
`A B C D parallelogram(5, 4, 105:)`
`draw(A, B, C, D)`
`mark(B, A, D)`
`mark(D, C, B)`
`mark(C, B, A, double)`
`mark(A, D, C, double)`
	

Codice Eukleides	Figura risultante

`% Somma vettoriale`
`A B C D parallelogram`
`draw(segment(A, B), full, arrow)`
`draw(segment(A, C), full, arrow)`
`draw(segment(A, D), full, arrow)`
`draw(segment(B, C), dotted)`
`draw(segment(D, C), dotted)`
	

`% Tangenti a una circonferenza`
`O = point(2, 2)`
`C = circle(O, 2)`
`A = point(6.5, 2)`
`c = circle(O, A)`
`I J intersection(C, c)`
`color(lightgray)`
`draw(line(A, I))`
`draw(line(A, J))`
`color(black)`
`draw(O, plus)`
`draw(A)`
`draw(C)`
`draw(c, dotted)`
	

`% Una spirale`
`frame(-5, -4, 5, 4)`
`trace(t, 0, 3*360)`
`{ point(t/360, t:) }`
	

`% Parabola con fuoco e direttrice`
`F = point(3, 1.5)`
`D = line(point(1, 0.5), -65:)`
`C = parabola(F, D)`
`draw(F)`
`draw(D)`
`draw(C)`
	

7.5 Altri esempi

Le figure presentate in questa sezione derivano dagli esempi inclusi nella distribuzione standard del pacchetto Debian.

Figura 7.53. Eukleides: metodo attribuito a Abdul al Wafa per la costruzione di un triangolo equilatero inscritto in un quadrato (il file `abdul_al_wafa_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

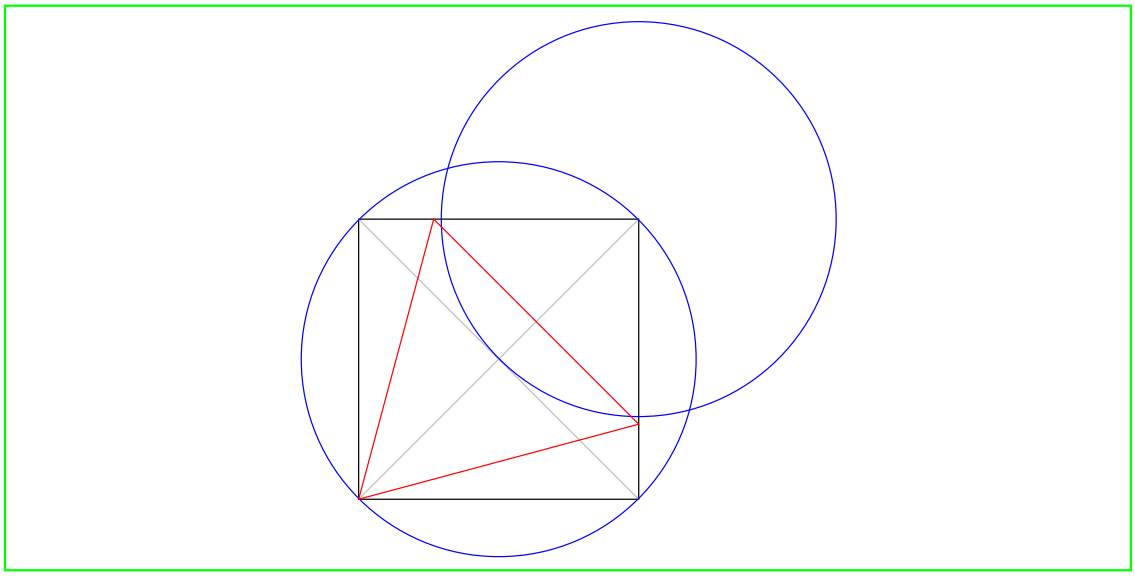


Figura 7.54. Eukleides: versiera di Agnesi (il file `agnesi_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

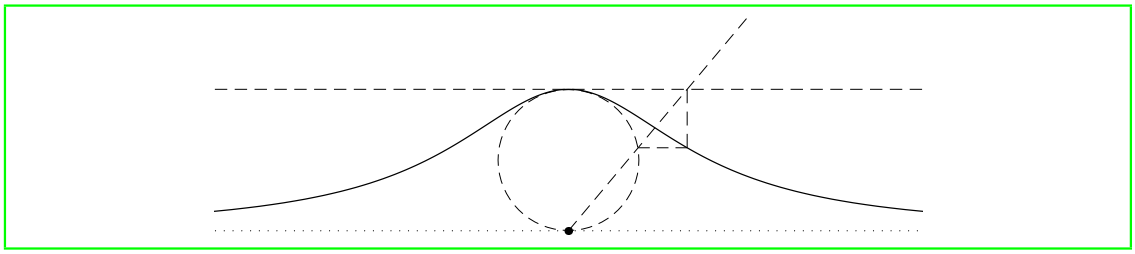


Figura 7.55. Eukleides: illustrazione per un esercizio sulla somma degli angoli α e β (il file `angles_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

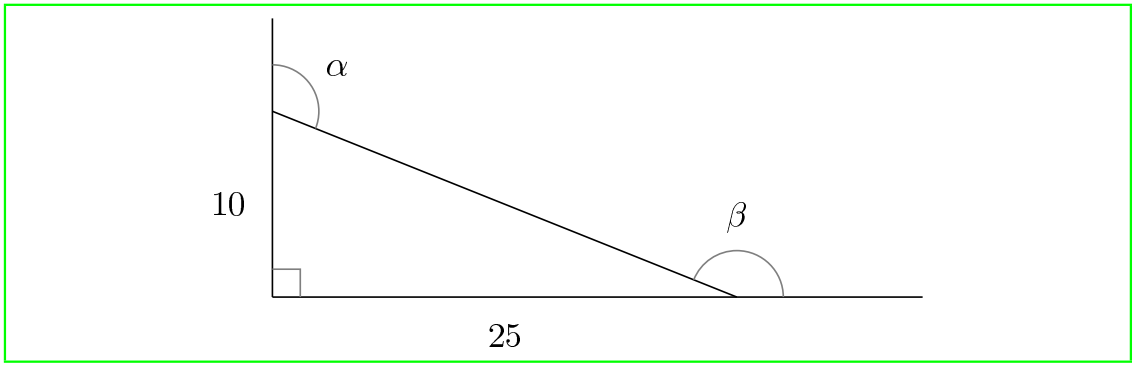


Figura 7.56. Eukleides: bisettrici in un parallelogrammo (il file `bisectors_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

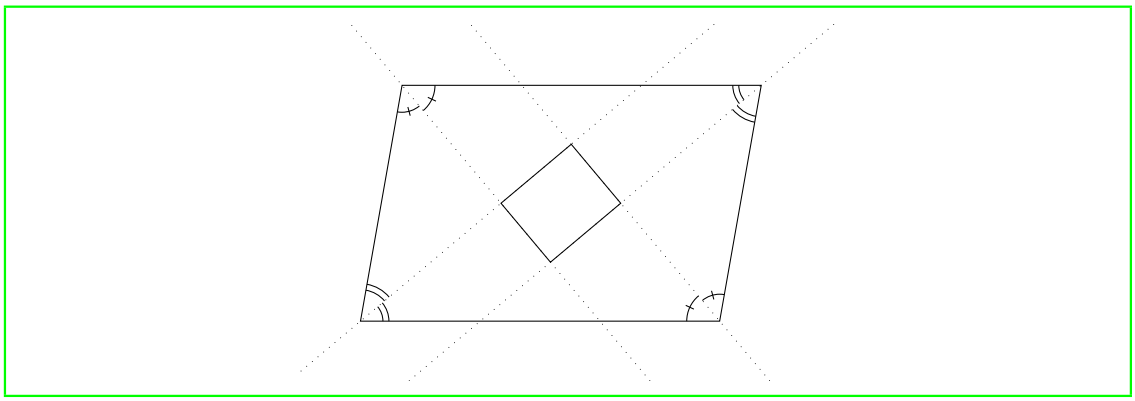


Figura 7.57. Eukleides: illustrazione per un esercizio sulla posizione del centro di massa dell'oggetto raffigurato al variare di r (il file `gravity_center_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

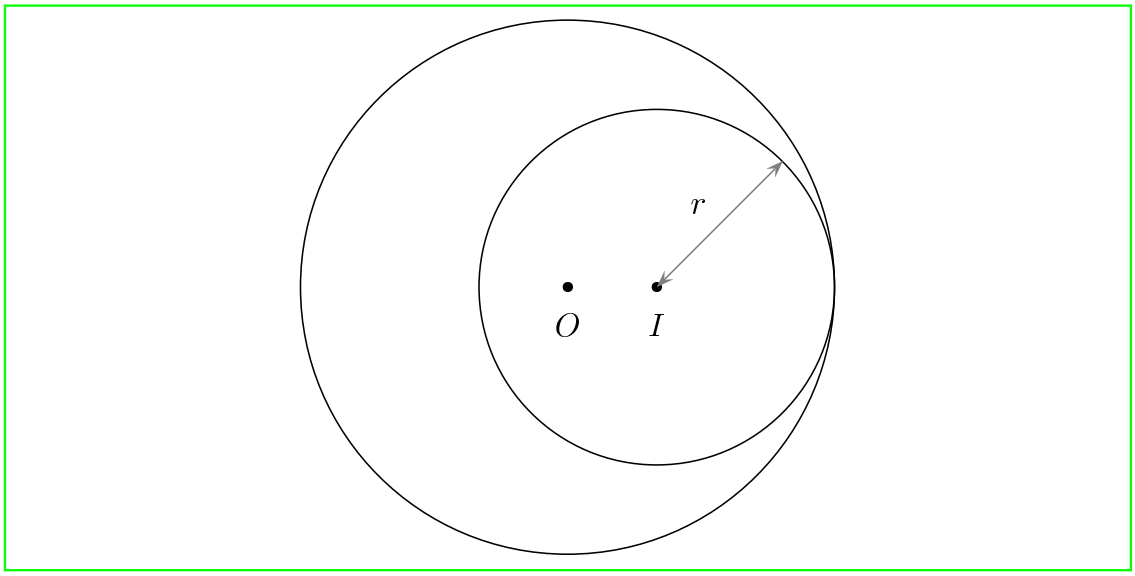


Figura 7.58. Eukleides: illustrazione relativa a un esercizio sulle omotetie (il file `intersection_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

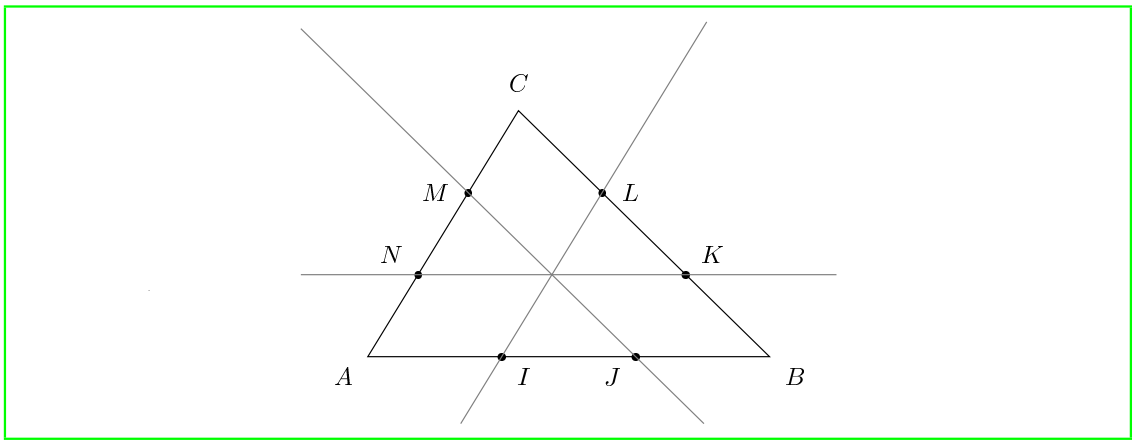


Figura 7.59. Eukleides: illustrazione relativa a un esercizio sulle omotetie (il file `lines_and_circles_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

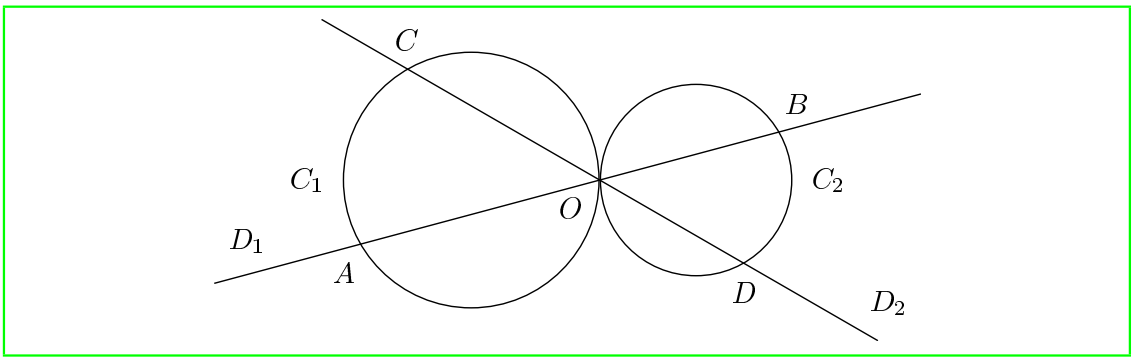


Figura 7.60. Eukleides: un classico problema sui luoghi geometrici (il file `locus_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

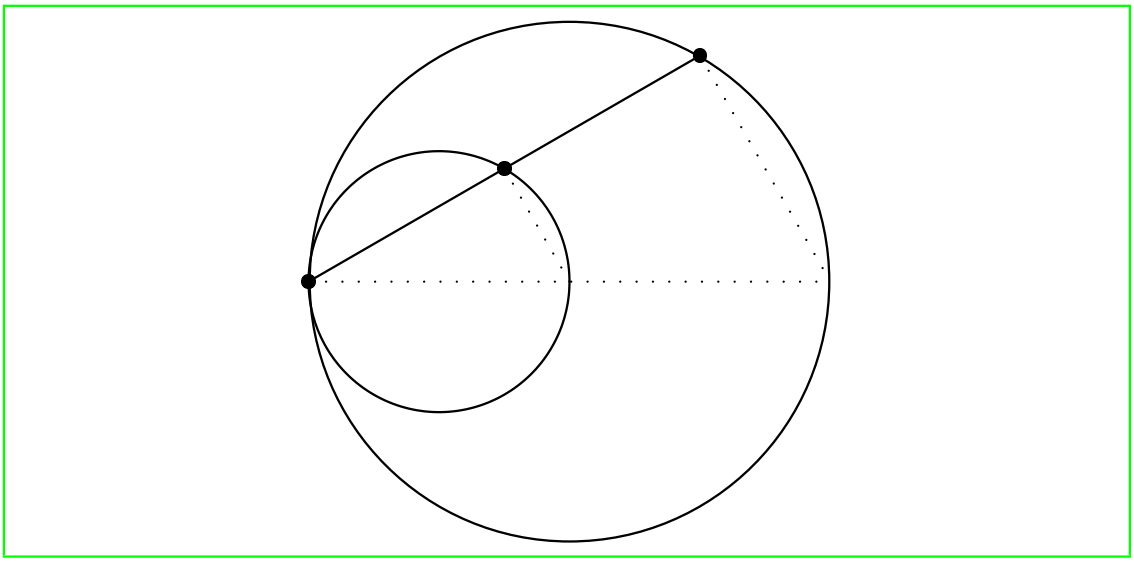


Figura 7.61. Eukleides: illustrazione del teorema di Morley (il file `morley_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

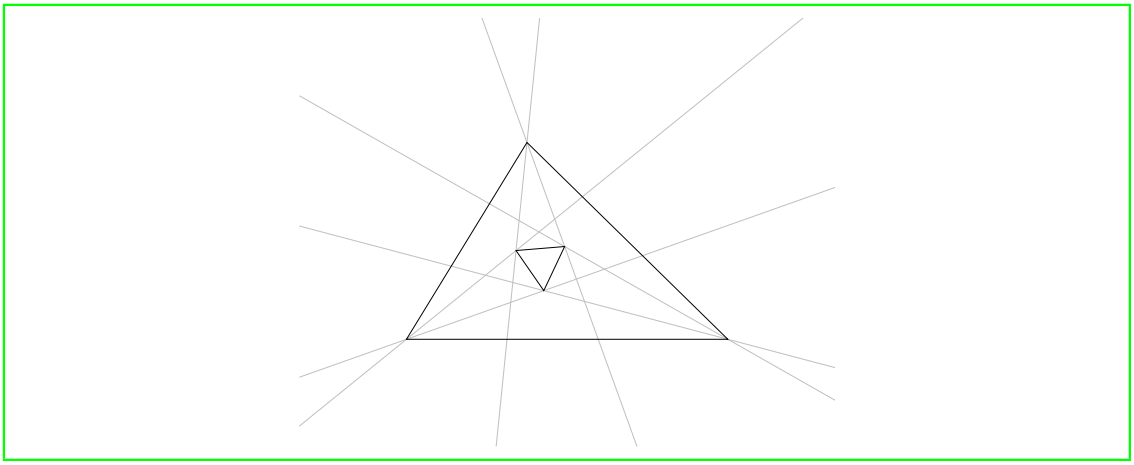


Figura 7.62. Eukleides: una proprietà dell'ortocentro (il file `orthocenter_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

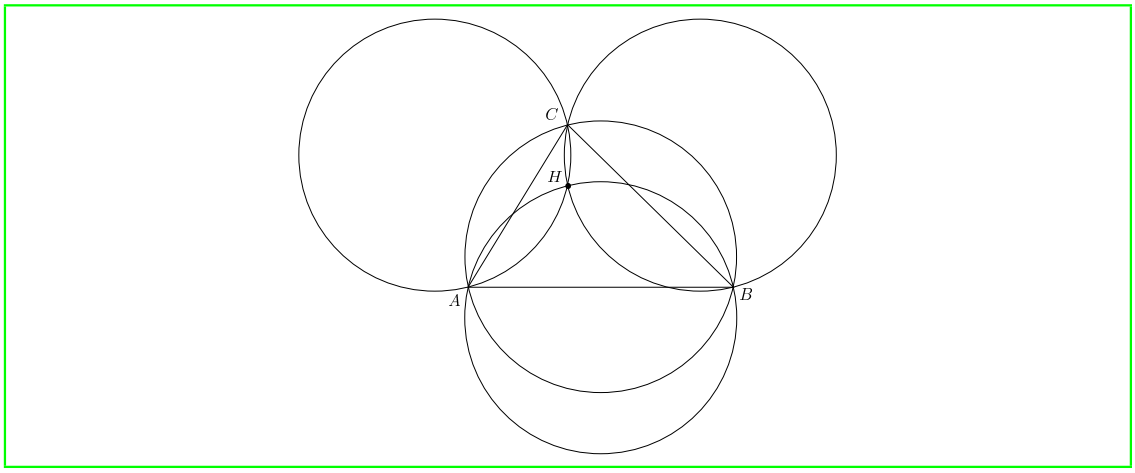


Figura 7.63. Eukleides: illustrazione relativa a un esercizio sulle omotetie (il file `parallelogram_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

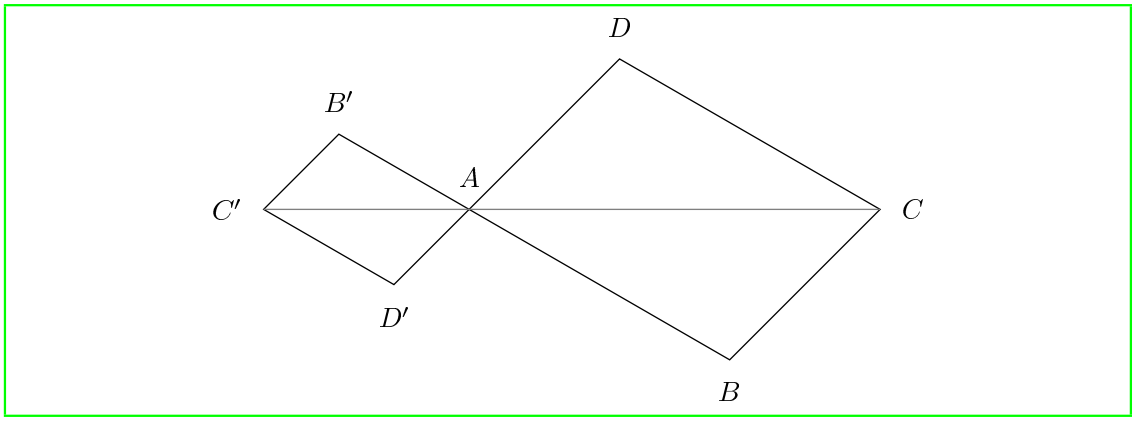


Figura 7.64. Eukleides: illustrazione del teorema di Pascal (il file `pascal_euk' contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

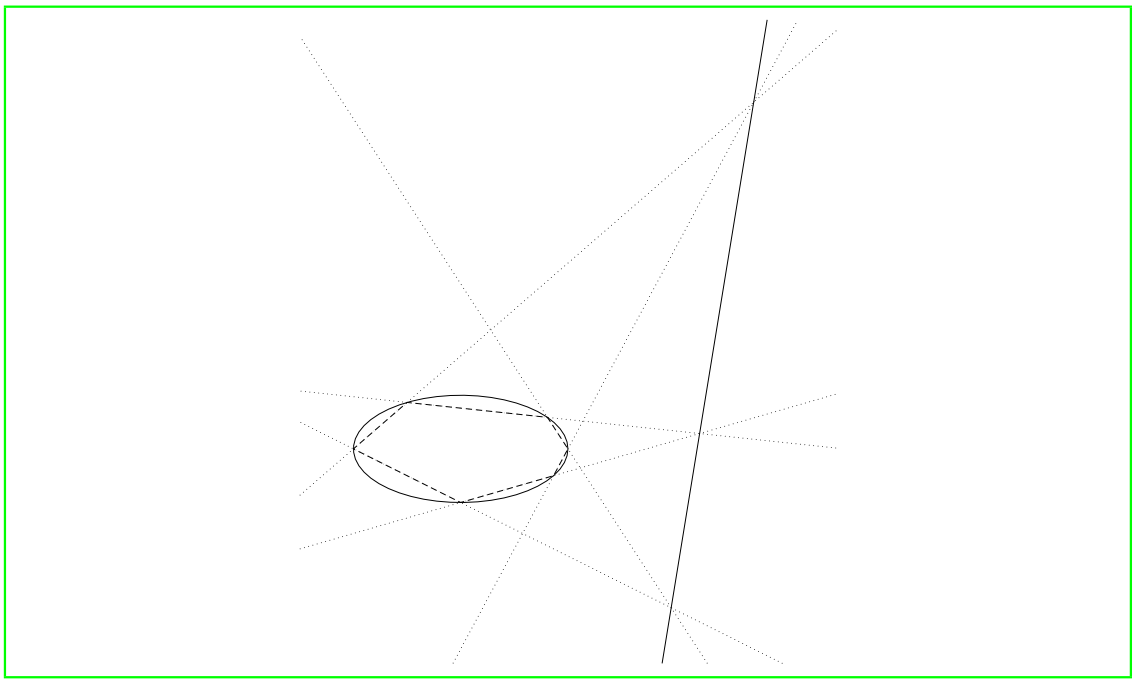


Figura 7.65. Eukleides: illustrazione per un esercizio sugli assi dei segmenti (il file `quadrilateral_euk' contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

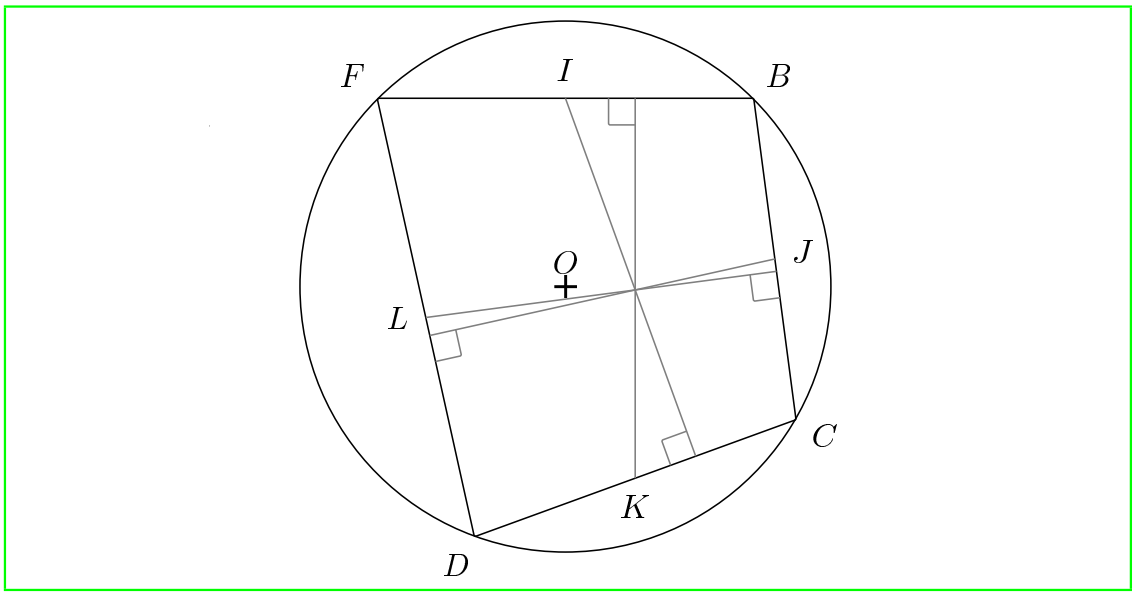


Figura 7.66. Eukleides: triangoli simili (il file `similar_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).

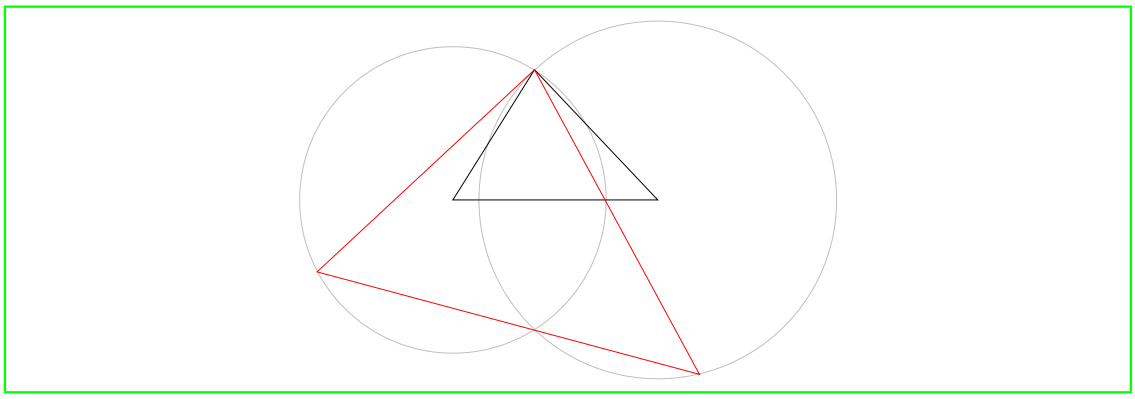
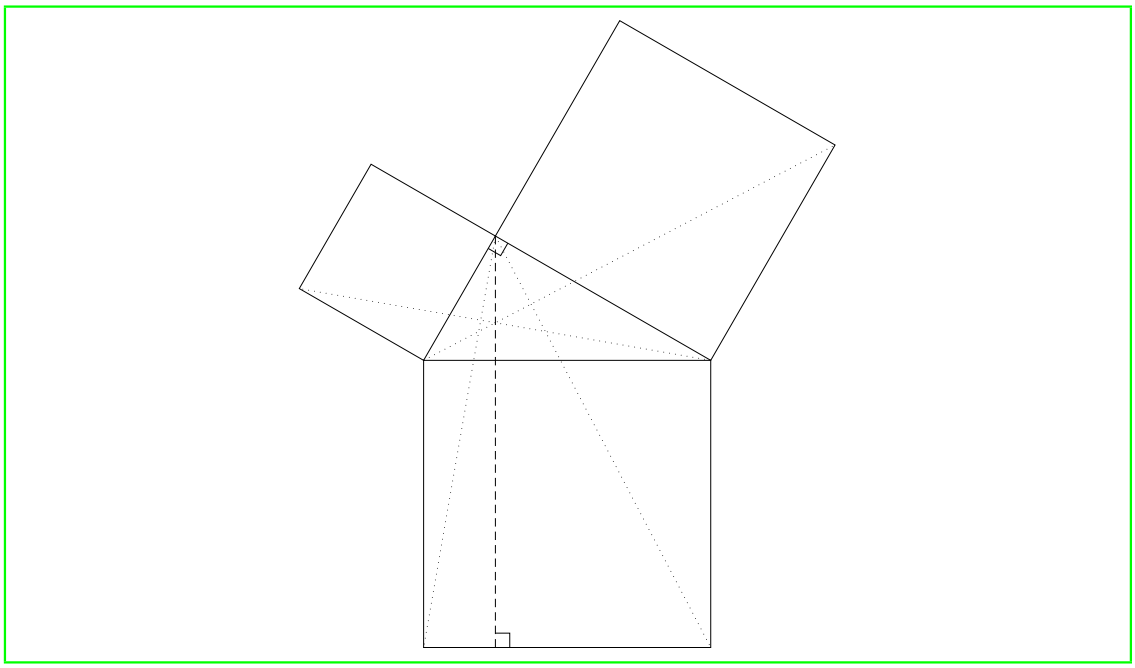


Figura 7.67. Eukleides: la classica figura di Vecten utilizzata da Euclide per dimostrare il teorema di Pitagora (il file `vecten_euk` contenente il codice Eukleides dovrebbe essere disponibile in allegato alla versione HTML del presente lavoro).



7.6 Riferimenti e approfondimenti

- Christian Obrecht, EΥΚΛΕΙΔΗΣ
  ⟨*http://www.eukleides.org/*⟩
- Christian Obrecht, *Eukleides: A geometry drawing language*
  ⟨*https://www.tug.org/TUGboat/Articles/tb22-4/tb72obre.pdf*⟩
- TUG, *Welcome to the PSTricks web site*
  ⟨*http://www.tug.org/PSTricks/*⟩
- X Consortium Inc., *X Logical Font Description Conventions*
  ⟨*http://www.xfree86.org/current/xlfd.pdf*⟩

- Bill Casselman
  - *Mathematical Illustrations*
    ⟨*http://www.math.ubc.ca/~cass/graphics/manual/* ⟩
  - *Pictures and Proofs*
    ⟨*http://www.ams.org/notices/200010/fea-casselman.pdf* ⟩
  - *Visual Explanations reviewed by Bill Casselman*
    ⟨*http://www.ams.org/notices/199901/rev-casselman.pdf* ⟩
- Alexander Bogomolny, *Pythagorean Theorem and its many proofs*
  ⟨*http://www.cut-the-knot.org/pythagoras/index.shtml* ⟩

---

[1] **Eukleides** GNU GPL

[2] Salvo quando diversamente specificato, la figura viene disegnata all'interno di un riquadro il cui vertice inferiore sinistro ha coordinate (-2;-2) e quello superiore destro coordinate (8;6).

[3] Quando si entra in modalità di presentazione il programma è nello stato "A"; essendoci 26 lettere da "A" a "Z" si possono definire 52 variabili interattive per la medesima figura (ogni stato può essere associato a due variabili, una per ciascuna coppia di tasti freccia); per cambiare stato basta premere il corrispondente tasto alfabetico.

[4] in gradi

[5] Usando '**`xeukleides`**', gli eventuali simboli '**`$`**' presenti nelle stringhe vengono ignorati, allo scopo di migliorare la leggibilità della presentazione; per '**`eukleides`**' invece tali simboli delimitano del codice TeX che cerrà compilato e inserito nella figura.

[6] in gradi

[7] All'inizio lo stato predefinito è "A".

[8] enfasi mia

[9] ... per non dire rudimentale: si invita il lettore volenteroso ad estenderne le funzionalità, magari utilizzando un altro linguaggio di programmazione.

Appendici

Appendice A

Informazioni aggiuntive sul software e altre opere citate

Alml, IV

GNU GPL

Eukleides, 120

GNU GPL

GNU Groff, 65

GNU GPL

GNU Plotutils, 47

GNU GPL

Jgraph, 102

GNU GPL

nanoLinux, IV

GNU GPL; i singoli applicativi sono sottoposti eventualmente alle loro condizioni specifiche

Python, 1

⟨*http://www.python.org/license.html*⟩

Indice analitico

Massimo Piai *(... circa 1975)*
⟨*pxam67 (ad) virgilio·it* ⟩